

Hybrid Modelling of Time-variant Heterogeneous Objects

DENIS KRAVTSOV

A thesis submitted in partial fulfilment of the requirements of
Bournemouth University for the degree of Doctor of Philosophy



July, 2011

This copy of the thesis has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and due acknowledgement must always be made of the use of any material contained in, or derived from, this thesis.

Abstract

Denis Kravtsov

Hybrid Modelling of Time-variant Heterogeneous Objects

The physical world consists of a wide range of objects of a diverse constitution. Past research was mainly focussed on the modelling of simple homogeneous objects of a uniform constitution. Such research resulted in the development of a number of advanced theoretical concepts and practical techniques for describing such physical objects. As a result, the process of modelling and animating certain types of homogeneous objects became feasible.

In fact most physical objects are not homogeneous but heterogeneous in their constitution and it is thus important that one is able to deal with such heterogeneous objects that are composed of diverse materials and may have complex internal structures. Heterogeneous object modelling is still a very new and evolving research area, which is likely to prove useful in a wide range of application areas. Despite its great promise, heterogeneous object modelling is still at an embryonic state of development and there is a dearth of extant tools that would allow one to work with static and dynamic heterogeneous objects. In addition, the heterogeneous nature of the modelled objects makes it appealing to employ a combination of different representations resulting in the creation of hybrid models.

In this thesis we present a new dynamic Implicit Complexes (IC) framework incorporating a number of existing representations and animation techniques. This framework can be used for the modelling of dynamic multidimensional heterogeneous objects. We then introduce an Implicit Complexes Application Programming Interface (IC API). This IC API is designed to provide various applications with a unified set of tools allowing these to model time-variant heterogeneous objects. We also present a new Function Representation (FRep) API, which is used for the integration of FReps into complex time-variant hybrid models. This approach allows us to create a practical multilevel modelling system suited for complex multidimensional hybrid modelling of dynamic heterogeneous objects. We demonstrate the advantages of our approach through the introduction of a novel set of tools tailored

to problems encountered in simulation applications, computer animation and computer games. These new tools empower users and amplify their creativity by allowing them to overcome a large number of extant modelling and animation problems, which were previously considered difficult or even impossible to solve.

Contents

Abstract	2
List of contents	4
List of figures	8
List of tables	15
Acknowledgements	16
Declaration	17
Abbreviations	18
1 Introduction	19
2 Related work	29
2.1 The boundary representation	29
2.1.1 The parametric representation	29
2.1.2 The polygonal representation	32
2.2 The representations for volumetric models	34
2.2.1 The voxel representation	34
2.2.2 Implicit surfaces	38
2.2.3 The Function Representation (FRep)	43
2.3 Heterogeneous objects modelling	53
2.3.1 Hybrid modelling	53
2.3.2 Implicit Complexes (IC)	56
2.4 A survey of computer animation techniques	62
2.4.1 Keyframe-based animation	63
2.4.2 Procedural animation	66
2.4.3 Conclusions	71
3 Theoretical framework for Dynamic Implicit Complexes	72
3.1 Motivation	72
3.2 Dynamic IC cells	73
3.3 Dynamic IC attributes	76
3.4 Extensions to FRep within the dynamic IC framework	78
3.4.1 Modelling domains	78
3.4.2 Space-time	81
3.5 Dynamic IC relations	85

3.6	Dynamic IC operations	90
3.7	Dynamic IC states and their components.	94
3.8	Dynamic IC instances	98
3.9	The IC-based model	102
3.10	Conclusions	107
4	Dynamic Implicit Complexes Framework: Technical Aspects and Tools.	110
4.1	Introduction	110
4.2	A description of the IC entities and their properties	111
4.3	Methodology of model definition	113
4.3.1	The high-level notation for the definition of an IC model	114
4.3.2	IC model definition using the notation	119
4.3.3	The process of model evaluation	125
4.4	The technical details of model evaluation	127
4.4.1	Dependencies and the order of evaluation	127
4.4.2	Handling an event requesting a modification of time	138
4.5	A brief description of the IC API	140
4.6	The FRep API as a subset of the IC API	145
4.6.1	HyperFun	146
4.6.2	Mapping FRep concepts to the FRep API	148
4.6.3	FRep API extensibility	158
4.6.4	FRep model manipulation	159
4.6.5	The FRep model format interchange	161
4.6.6	The FRep library	165
4.6.7	Integration of FReps into the IC framework	168
4.6.8	Performance	170
4.7	Conclusions	174
5	Applications and results	176
5.1	An introductory 2D dynamic IC model exemplar	176
5.2	An exemplar of a 2D dynamic IC model with dependencies	184
5.3	Multidimensional dynamic models in the space-time domain	193
5.3.1	Introduction to space-time blending	193

5.3.2	The application of affine transformations in the space-time domain.	194
5.3.3	Additional time-dependent deformations.	196
5.3.4	Non-linear sampling in the space-time domain.	198
5.4	Implicit “Stand-ins”: a case of time-variant hybrid modelling	201
5.4.1	Introduction to the “stand-ins” technique	201
5.4.2	Background	203
5.4.3	Problem statement and approach outline	205
5.4.4	The proposed approach	207
5.4.5	The IC model definition	211
5.5	The controlled metamorphosis of animated meshes	215
5.5.1	Introduction	216
5.5.2	Method Outline	216
5.5.3	Conclusions	221
5.6	The “Andyhausen” experiment	221
5.6.1	Model overview	221
5.6.2	Model description	225
5.6.3	Conclusions	230
5.7	An interactive modeller for the definition of volumetric hybrid models	232
5.7.1	General description	232
5.7.2	Implementation overview	239
5.7.3	Future work	240
5.8	Conclusions	243
6	Conclusions and future work	246
6.1	Contributions	248
6.2	Future work	251
	References	253
A	A detailed description of IC entities and their components	270
B	IC model description of a “Stand-in” case study	281
C	IC description of “Andyhausen” experiment	287

List of Figures

1	An example of polynomial curve passing through a set of control points.	30
2	A patch of a parametric surface.	31
3	The components of a polygonal model.	32
4	Voxel data.	35
5	Volumetric data stored as discretized distance field (a) Two orthogonal slices of distance data (b) Volume rendering of volumetric data with colour variation.	37
6	An example of adaptively sampled distance field in 2D and the underlying adaptive data structure (Frisken <i>et al.</i> , 2000).	38
7	A scalar field (defining function): (a) The sign of a scalar field, (b) The extracted implicit surface ($T=0$) and (c) Different iso-surfaces for different values of T	39
8	An example of an implicit object.	40
9	An example of a CSG tree.	41
10	Set-theoretic operations based on R-functions: (a) Union and (b) Intersection.	46
11	The constructive Hypervolume model: (a) The original geometric object, (b) A visualisation of the space partitions and (c) The resulting Hypervolume object.	51
12	A set of cells present in the IC.	60
13	The topological relations of the IC.	60
14	A set of keydrawings reflecting important stages of a walk-cycle (image courtesy of Jose Fonseca).	64
15	A set of key-poses defined by the computer animator (“Andy” model courtesy of John Doublestein).	64
16	A set of cells present in the IC.	75
17	The motion of the disk defined over time.	76
18	A space mapping used for attributes: (left) initial geometry and attributes; (centre) deformed geometry and initial attributes (right) the same deformation applied to geometry and attributes simultaneously.	77
19	The “reduce dimensionality” operation.	81

20	Meeting the requirements of a 3D Cartesian product.	82
21	Space-time object: (a) Two defining 2D projections (b) A set of 2D geometric slices of the resulting space-time object. . .	83
22	A set of 2D cross-sections of the objects metamorphosed over time.	84
23	A 3D objects defined in the space-time domain.	84
24	The dependency relation between the parameters of two cells.	89
25	The modification of the radius depends on the translation of the rectangle.	90
26	Example definition of a simple deformation.	91
27	Example definition of an operation with constant parameters.	93
28	Example definition of an operation with time-dependent parameters.	93
29	Example definition of an operation with time-dependent parameters and time-dependent operands.	94
30	Transitions between the states of the cell through reactions issued in response to external events and a set of generated events.	97
31	Transitions between the IC instances of the IC-based model.	100
32	Dynamics of transitions between the structural states of the model over time.	101
33	The structure of the dynamic IC model and its components. .	106
34	The topological relations of the IC.	106
35	The set of cells at an intermediate modelling phase.	107
36	The set of cells at an intermediate modelling phase.	108
37	A set of instances reflecting the model at different time-spans.	108
38	A high-level UML diagram of IC entities.	112
39	The structure of the textual IC model definition.	115
40	The syntax diagram for the definition of entities present in the model.	116
41	The syntax diagram for the definition of reactions to events. .	117
42	The syntax diagram for the definition of internal components of the cell.	118

43	The syntax diagram for the definition of template ICs and IC instances.	119
44	The initial set of cells and the dependencies between them.	129
45	The lists of topologically sorted cells.	129
46	The list of concurrently evaluated cells.	130
47	A simple model of Newton's cradle.	132
48	The four states of Newton's cradle.	132
49	The two states of the cradle: (Left) One of the balls moving, no collision (Right), One of the balls moving (collision situation).	133
50	A simple bi-directional dependency.	134
51	Two different evaluation orders.	135
52	Two different user-controlled evaluation orders.	135
53	The set of cells and dependencies between them.	136
54	Two different user-controlled evaluation orders.	136
55	The dependencies between the properties and the dependency graph between the cells.	138
56	On-demand adjustment of time (a) Time-step back in time (b) Going through all the previous states.	139
57	The structure of the IC API and related tools.	141
58	The UML diagram of the main classes present in the IC API.	142
59	The IC modelling work flow.	144
60	Full structure of the modelling environment and basic tools for IC modelling.	144
61	A simplified UML diagram of an FRep entity	149
62	The high-level overview of the types of FRep entities.	150
63	Types of primitives available in the FRep API.	151
64	Types of entities available in the FRep API.	152
65	The difference between a primitive and an operation.	153
66	The dynamic diagrams illustrating the evaluation of different types of entities.	154
67	The FRep entity UML-diagram.	156
68	The first phase of the FRep tree evaluation.	157
69	The second phase of the FRep tree evaluation.	158

70	The interaction with the FRep API.	160
71	A Model-View-Controller diagram: (a) The general design pattern (b) The MVC and FRep API	161
72	The entity serialiser.	162
73	The FRep entity and its XML definition.	164
74	The generation of the FRep API related components from an entity description.	167
75	The generation of application specific code required for the integration of FReps.	168
76	A set of application and platform specific translators.	168
77	The integration of the FRep entities into the IC API.	169
78	The evaluation of the F-cell within the IC framework.	170
79	A set of cells present in the IC.	177
80	The motion of the disk defined over time.	178
81	The topological relations of the IC.	179
82	The second IC instance.	180
83	The third IC instance.	182
84	A dynamic IC over time (here the time parameter of the IC is its local time).	184
85	The cells and relations of the dynamic IC.	184
86	The cells and the dependency relations between them.	185
87	The first IC instance.	187
88	The second IC instance.	187
89	The third IC instance.	189
90	The fourth instance.	190
91	The cells and relations of the fifth IC instance.	192
92	The IC instances and the transitions between them.	192
93	Space-time blending: (a) Two initial 2D objects (b) A set of intermediate objects generated using space-time blending.	193
94	The IC cells initially present in the model.	194
95	The result of the application of a space-time blending operation: (a) Regular space-time blending (b) The proposed space-time blending with an additional affine transformation.	195

96	The cross-sections of the shape generated using the improved space-time blending: (a) a user guided rotation around the time axis to align object features (b) A user guided scale along the time axis.	195
97	The transition between 3D objects (a) Linear metamorphosis (Pasko <i>et al.</i> , 1995). (b) Improved space-time blending. . . .	196
98	Examples of transitions using space-time blending for 3D objects.	197
99	Problems caused by disjointed components appearing during the transition: (a) Regular space-time blending (b) Space-time blending with additional deformations.	197
100	The dependency relations for an improved space-time blending with additional deformations.	199
101	Examples of the extracted “thick features” (marked by crosses).	199
102	Animated mesh information: (a) Polygonal mesh, (b) Rigging skeleton, (c) Skinning information. Model ”Andy” courtesy of John Doublestein.	206
103	Initial approximation: (a) The initial placement of bounding volumes inside the mesh, (b) The shape produced by convolution surface.	207
104	The synchronised motions of the embedded convolution surfaces during the animation process.	209
105	Phases of interaction between animated objects without blending (left) and with blending (right): (a) Two implicit surfaces and a single blend shape during blending, (b) The boundary case before the two shapes separate, (c) Two separate shapes with some deformation showing the objects’ reciprocal attraction.	211
106	Viscosity: (a) low, (b) medium, (c) high.	211
107	Proposed approach outline.	212
108	The different states of the model.	212
109	The topological relations of the model.	213
110	The dependency relations of the model.	214

111	A set of examples of the dynamic hybrid modelling technique.	215
112	All IC instances of the controlled metamorphosis example together with the dependency relations.	217
113	The different stages of the projection of the BRep mesh to the FRep shape of the “stand-in”: (a) Simplified example (b) Projection of a character’s head to the appropriate “stand-in”.	218
114	The controlled metamorphosis of animated meshes: (a) Jumping girl to a running zebra metamorphosis (b) Crawling monster to a levitating robot metamorphosis.	220
115	The different states of the model.	222
116	The cells initially present in the IC.	222
117	The first IC instance.	223
118	The boundary relations of the first IC instance.	224
119	The containment relations of the first IC instance.	224
120	The dependency relations of the first IC instance.	224
121	The boundary relations of the second IC instance.	228
122	The containment relations of the second IC instance.	228
123	The dependency relations of the second IC instance.	229
124	The third IC instance of the model.	229
125	The boundary relations of the third IC instance.	230
126	The dependency relations of the third IC instance.	231
127	The FRep shelf added to Maya.	233
128	The FRep proxy object representing a solid sphere in Maya.	234
129	Constructive FRep tree shown using the Maya Dependency Graph or Hypergraph.	234
130	Two rendering options in Maya: (a) The rendering of proxy primitives is enabled, (b) The rendering of proxy primitives is disabled.	235
131	The parameters available for the blending union operation. .	235
132	Animating the available parameters of FRep entities using the Maya animation tools.	236
133	Example of an FRep model created within Maya shown from different angles.	236

134	An overview of the XML description of the model shown in figure 133.	237
135	The FRep tree of the model shown in figure 133.	238
136	Examples of volumetric models produced using our modeller.	239
137	The FRep castle model: (a) Using the regular Marching Cubes algorithm, (b) Using the Marching Cubes algorithm with post processing extracting sharp features.	241
138	Screenshots of the working environment: (a) The definition of the “Stand-in” model, (b) The setup of the improved space-time blending model, (c) Exploring the FRep tree contained in the IC F-Cell used for space-time blending.	242
139	Scanned voxel data of a patient blended with FRep entities.	243
140	Examples of rapidly prototyped (through 3D printing) FRep objects defined using the HyperFun package.	244

List of Tables

1	The list of available FRep modelling domains.	79
2	A list of dynamic IC entities and their properties.	278
3	The components of the animation entity.	280

Acknowledgements

First of all I would like to express my gratitude to my supervisors Prof. Peter Comninos, Prof. Alexander Pasko and Dr. Valery Adzhiev for their continued support and advice. I would also like to thank them for their help in improving the readability of this document.

I would like to thank Oleg Fryazinov (NCCA) for his help and advice, Elena Kartasheva (Russian Academy of Sciences) for useful discussions and her contribution to the theory of dynamic ICs, Adam Vanner (NCCA) for sharing his expertise in the latest techniques used in modern animation systems, Andreas Baerentzen (Technical University of Denmark) for the discussion regarding voxel representations and various approaches to volumetric modelling, Jose Fonseca (NCCA) for providing the information regarding traditional animation and computer animation techniques and John Doublestein (Savannah College of Art and Design) for making the “Andy” character model publicly available.

Thanks also goes to the National Centre for Computer Animation and the Media School at Bournemouth University for the scholarship, which made it possible for me to undertake this research.

Last but not least I would like to thank my family, friends and fellow students for their support during the years of my intense work on this project.

Declaration

This thesis has been created by myself and has not been submitted in any previous application for any degree. The work in this thesis has been undertaken by myself except where otherwise stated. The materials related to hybrid modelling technique based on “Implicit Stand-ins” have been published in (Kravtsov *et al.*, 2010b,a,c). The work regarding the improvements of space-time blending appeared in (Pasko *et al.*, 2010).

Abbreviations

API	Application Programming Interface
BRep	Boundary Representation
CAD	Computer-Aided Design
CPU	Central Processing Unit
CSG	Constructive Solid Geometry
EM	Empirical Modelling
FRep	Function Representation
GPU	Graphics Processing Unit
IC	Implicit Complexes
LLVM	Low Level Virtual Machine
MVC	Model-View-Controller
PRep	Parametric Representation
UML	Unified Modelling Language
XML	Extensible Markup Language

1 Introduction

The physical world consists of a wide range of miscellaneous objects of a diverse nature. Models of these real entities can help us gain a better understanding of the physical world.

In the past a lot of research was focused on modelling simple homogeneous objects made of a uniform material. This research effort resulted in the development of a number of advanced theoretical and practical methods for describing physical objects. Specific mathematical representations, theoretical frameworks and modelling tools have been introduced over the years. As a result, the process of modelling and animation of certain types of homogeneous objects became easier to do. However, not all physical objects can be described as homogeneous objects.

In fact, the majority of physical objects are heterogeneous in nature and it is necessary to be able to work with such objects. Heterogeneous objects are composed of different materials and have a complex internal structure. For instance, a walnut consists of a shell and a seed contained in the shell. In its turn the shell has a number of inner layers and the seed has a complex internal structure as well. Complex assembly of a set of dynamic objects made of homogeneous matter can also be considered a heterogeneous object. Heterogeneous object modelling is a very promising approach, which is likely to be useful in a wide range of applications. This is still a new and evolving research area. New specialised representations and theoretical frameworks are being introduced and existing ones are still being refined. Despite all the potential advantages of this new approach, there are no existing tools, which would allow us to work with static and dynamic heterogeneous objects. At the moment we can only conclude, that the heterogeneous nature of the modelled objects makes it appealing to employ a combination of different representations using a hybrid model.

The latest advances in computer hardware resulting in an increase of computational power, the introduction of modern 3D displays and of new types of haptic devices make heterogeneous object modelling feasible. Recent research has presented a number of ways allowing us to perform this type of

modelling. In this thesis we present a new framework incorporating a number of existing representations and animation techniques. This framework can be used for the modelling of dynamic multidimensional heterogeneous objects. We believe that the new tools built upon this framework will prove to be useful in various areas of computer graphics.

The Implicit Complexes (IC) Framework, developed earlier by our research team¹, provides us with a way of integrating models of different nature within one hybrid model by combining both the geometry of objects and their arbitrary properties. To date however, this framework has only been suitable for the modelling of static heterogeneous objects. However considering the walnut example in more detail, we notice that its properties change over its lifetime.

Most natural objects are not static but undergo certain modifications or transitions over time. More importantly, these dynamic heterogeneous objects can interact with each other over time in an unpredictable number of ways. In general, the internal properties of time-variant objects may depend on the properties of various external objects. Some of these interactions could be defined manually or by using a higher-level procedural definition, while certain behaviours could only be determined as a result of a simulation process.

In this thesis we describe a new IC-based framework, allowing us to model time-variant multidimensional heterogeneous objects using the aforementioned approaches. This dynamic IC framework provides a way of defining the objects using a number of existing representations and animation techniques. Here we present a brief description of the existing representations and animation approaches that are combined in the new dynamic IC framework in order to provide a set of tools for the definition of complex dynamic hybrid models.

The modern world of computer graphics is mostly dominated by boundary representation models (also known as BRep models). Such models can only store information about an object's boundary (as though the object was hollow). An object's boundary information alone is sufficient for a wide range of applications, such as certain types of computer animation, a large set of

¹The list of collaborators includes Elena Kartasheva, Valery Adzhiev, Alexander Pasko, Peter Comninou, Oleg Fryazinov and Benjamin Schmitt.

computer games and even for a limited subset of CAD modelling applications or physical simulations. One of the most popular types of boundary representation models are the polygonal models. With these models planar primitives such as triangles or quads are used as building blocks to represent 3D objects. Polygonal models provide a rich set of available operations and are often highly scalable. One of the most important reasons why polygonal models have gained such popularity in the past thirty years is the fact that planar primitives can be rendered in a relatively easy way. This was quite an important factor in the early years of computer graphics when existing hardware resources were very limited. The extensive development of computer graphics hardware was mostly oriented towards maximising the number of planar primitives rendered per frame and the introduction of new rendering techniques that could somehow enhance the visual quality of the resulting images.

As was mentioned earlier, boundary representation models are only suitable for a limited set of applications. Naturally when the internal structure of an object is required the limitations of the boundary representation become apparent. For instance, the BRep description of a detailed walnut model, mentioned earlier in the text, would be very limiting. The boundary representation would only allow us to create a finite set of surfaces representing the shell and the seed contained within it. It would be next to impossible to explore the internal structure of such an object, if we were to cut or split the nut, attempting to look at its contents hidden under the shell. A detailed model of such an object requires a more powerful volumetric representation. A volumetric representation allows us to describe the surface as well as the interior of some region of space. Thus, volumetric models allow us to overcome the aforementioned limitations of the boundary representation. The volumetric representation is more natural and provides us with the ability to store more detailed information about an object's interior, which is especially important when dealing with heterogeneous objects (i.e. objects consisting of different materials and having complex internal structures). The volumetric representation, unlike the boundary representation, affords us more freedom in terms of our ability to design, explore and manipulate the model.

One of the many types of volumetric representations relies on discrete

voxels, which are a 3D equivalent of pixels. Models of this type are able to store significantly more detailed information about the represented objects and their relations without the familiar restrictions found in BRep models. But the main problem with voxels is that they are resolution dependent. It is thus desirable to provide a model of the highest possible resolution to avoid major aliasing artefacts. Increasing the model resolution leads us to another major problem with voxel representations: the significant storage requirement caused by the necessity to store the information relating to the entire discrete volume set. There are different techniques allowing us to overcome this issue, but they are not well suited for dynamic models.

There is another volumetric representation that we could employ. A Function Representation (FRep) is a generalised model representation allowing us to define solid objects of arbitrary dimensionality and to mix objects of different dimensionalities within a single model. An FRep system comprises of a set of geometric primitives, a set of operations and a set of relations. Primitives are combined using operations. An FRep usually allows us to represent compactly advanced volumetric models in a resolution independent manner. The resolution independence of the model is a rather important factor, meaning that the model can easily be refined depending on the specific application needs. A constructive tree used in combination with an FRep makes it easier to see how the model was assembled and to modify the model after it was constructed. This is vital not only for static models but also for dynamic models, because it affords us the freedom to introduce dramatic changes to the geometry and topology of the model. It is also important to note that an FRep can be used to specify the arbitrary volumetric attributes of the modelled objects so as to represent their internal structure in different contexts. One of the significant shortcomings of FRep models is their computationally expensive model evaluation procedure. Another issue is their limited support in current software tools and the absence of certain methods used in existing computer animation systems, which can limit the use of FReps in a number applications.

Overall, we can see that the various model representations have distinct advantages and disadvantages.

We have mentioned a number of issues specific to BRep models, but BRep models cannot easily be replaced by volumetric models for a number of reasons. A large number of techniques for modelling and animation of BReps was developed over the years. A wide range of applications and tools relying on BRep models was introduced during the past thirty plus years. There is an immense amount of content produced by a significant number of professionals trained to work with BRep models alone. Large groups of experts specialising in certain ever-narrowing areas of static or dynamic BRep modelling do not have enough experience to work efficiently with other representations. Besides, as was mentioned earlier, BRep models satisfy all the needs of many existing applications. In these cases the definition of a more generic models would be unnecessary. It is thus apparent, that the complete rejection of BReps as well as other existing model representations would be inappropriate. The necessity to introduce a generic hybrid modelling system allowing us to combine objects of various representations thus becomes apparent. Such a unified representation should allow us to exploit the best characteristics of the various representations and of the ways in which objects expressed in these diverse representations can be made to interact with each other.

The Implicit Complexes (IC) Framework allows us to combine models of diverse nature within one hybrid model. ICs were introduced as a representation for a cellular-functional hybrid model of heterogeneous objects. The hybrid model described within ICs may contain entities of various dimensionalities and representations. IC models consist of valid topological descriptions of heterogeneous objects and allow for the combination of different existing representations of both the geometry of objects and the attributes describing the objects' properties. This framework provides the user with a set of powerful tools. Unfortunately, up to date the IC framework was only suited for the definition of static hybrid models. Thus, it could only be used for a limited set of applications.

As we have mentioned earlier in the text, the capability of defining time-variant models is very important for an appropriate description of a large variety of applications. A large number of the existing animation systems provide the user with a wide range of tools suitable for the definition of dynamic models. But the majority of these systems are oriented towards one representation

only. Most commonly, these systems support BRep models alone, as they are mostly targeted to the visualisation or interaction with the object's boundary. Thus, it becomes apparent that in modern animation systems there is a necessity for representing objects of different types that coexist in the same scene and interact with each other in a seamless way.

There are several techniques commonly used in the production of animation sequences, such as keyframing and inbetweening. These techniques are similar to the ones used in traditional hand-drawn animation, making them more accessible to users with an artistic background. These approaches provide the artist with precise control over various aspects of the produced animation sequence. Unfortunately, this workflow can still be a very arduous and time-consuming process. Besides, major modifications of the animation sequence require a lot of manual repetitive actions from the animator.

Another way of defining a computer animation sequence is through a script (also known as a procedural definition). Scripting in some sense can be seen as a rather detailed breakdown of a screenplay. This description can then be used by the animation system to produce the actual frames of the animation sequence. This approach also allows the user to define the model and the processes taking place within it in an appropriate way, without the necessity of describing all the aspects manually. In certain circumstances, the complex evolution of a model can be more easily described through a scripting language incorporating high level terms used for animation production. This approach is also better suited for the definition of real-life processes, when the animator is more interested in the correctness of a model. The valid definition of a model can help the user produce a believable animation sequence reflecting the desired behaviour of dynamic entities. In contrast, a similar sequence animated by hand (which may not always be possible) would only try to laboriously reproduce a certain phenomenon aiming at visual resemblance without a good understanding of the dynamic process. Compared to a procedural definition, keyframing leaves little room for subsequent modifications and extensions. A procedural definition on the other hand allows the user to build a model iteratively, combining different objects and states of the model together, and thus creating a model with a behaviour that may not be known in advance. Another powerful feature of this approach is the possible pres-

ence of dependency relations between the entities of the model. This allows the user to define a set of independent agents and their behaviour in a modular fashion and to combine those, establishing dependencies between them. One of the major issues of this approach is the high level of technical skills required of the person developing the procedural definition of the model. Many artists find this approach rather hard to understand and difficult to direct in a predictable way.

It is hard to give preference to any one of these methods as they are suited to the solution of dissimilar problems. Keyframed animation is often easier for artists to work with, while a procedural definition, which can be thought of as a form of computer programming, may be much easier for technical people collaborating with artists. Additionally, some low-level animation sequences often have to be defined manually before being incorporated into the procedural definition of a model. Certain complex models can only be defined in a procedural way, as they would otherwise require a significant amount of laborious and repetitive work. The combination of these approaches allows us to get the best of both worlds, having precise artistic control over some parts of the model, while providing a high-level definition of other parts of the model that can be evaluated on the fly.

We can see that over the years a wide range of different techniques have been introduced for the creation of time-dependent models. The situation in the world of animation is similar to the one we described in relation to the modelling of static objects. Each of the approaches has its distinct advantages as well as disadvantages. In this thesis we describe a new framework allowing us to combine the aforementioned animation techniques for the hybrid modelling of time-variant heterogeneous objects. We provide a way of describing heterogeneous objects defined using different modelling techniques involving boundary and volumetric representations as well as their procedural variations. Interactions between objects of different representations defined within the framework can take place seamlessly. The introduction of these new capabilities requires the development of new mathematical structures able to represent diverse models - specifically in the context of computer animation and simulation. A multilevel modelling system based on this mathematical framework allows us to create a set of practical tools tailored for

complex multidimensional hybrid modelling of dynamic heterogeneous objects. These new tools will allow us to overcome a large number of existing modelling problems and to provide the user with a number of new powerful tools previously unavailable.

This thesis is structured as outlined below.

In Chapter 2, we present related work and we briefly describe existing model representations. We discuss the advantages and disadvantages of the various representations together with their application areas. The IC framework is introduced as a common platform allowing us to incorporate all the representations within one hybrid model. In the second part of this chapter we provide a survey of existing animation techniques and methods used for the definition of time-dependent models. In similar way to the discussion regarding existing model representations we analyse the strengths and weaknesses of the existing approaches for the definition of dynamic models. At the end of the chapter we discuss the necessity for the extension to the IC framework in order to use it for the definition of time-variant heterogeneous objects.

In Chapter 3, we discuss the set of extensions required for the Dynamic IC framework. We provide extended definitions of the basic concepts and we introduce the new notions required for the dynamic modelling of complex time-variant heterogeneous objects. In this chapter we also discuss various components of the framework, paying particular attention to the Function Representation and the Constructive Hypervolume Framework. A number of extensions to these frameworks are described in this chapter. These extensions provide us with a number of new tools useful for multidimensional hybrid modelling.

In Chapter 4, we present a more detailed description of the IC framework from a technical point of view. We describe the high-level notation used for the definition of the models together with the methodology of model definition. We outline the process of model evaluation with respect to complex dynamic relations established between the objects. This information is required for the practical implementation of the framework. Further, in this chapter we introduce the Implicit Complexes Application Programming Interface (IC API). The IC API is designed to provide various applications with

a unified set of tools allowing them to perform modelling of time-variant heterogeneous objects. In the remainder of the chapter we discuss the problems of integration of the existing representations into the framework. A practical implementation of the whole framework is outside the scope of this research project. Thus we mainly focus on the integration of FReps into the framework. Hence, the design and the implementation of the FRep API are described in more detail in this chapter as well. At the end of the chapter we consider further improvements of the APIs and we discuss a number of ways to improve the performance of model evaluation.

In Chapter 5, we describe a number of applications of the proposed modelling framework and discuss the results. We outline a set of proposed improvements for space-time blending relying on multidimensional dynamic models. Next we introduce our new method for the modelling of interactions between dynamic objects and viscous materials using a time-dependent hybrid models. We also describe the extensions to this approach which allow us to solve a number of other existing problems, including the partial metamorphosis of animated characters and the controlled metamorphosis of dynamic meshes. Then we present an application involving a complex interaction sequence between a set of time-variant heterogeneous multidimensional objects within one hybrid model. Finally, we describe our prototype implementation of an interactive modelling system that can be used for the definition of dynamic heterogeneous objects or certain parts of a hybrid model. In addition, we describe in detail practical methods that can be employed for the acceleration of model evaluation employing both the CPU and the GPU.

In Chapter 6, we examine the results of the presented work, discuss potential solutions of the problems that have been investigated and outline our conclusions. At the end of this chapter we summarise the contributions of our work and propose directions for future work.

The main contributions of this research work are:

1. The introduction of a new dynamic IC framework.
2. The introduction of the new extensions to the Constructive Hypervolume Framework.

3. The design and detailed description of a new high-level language for the definition of time-dependent mixed-dimensional hybrid models.
4. The design and implementation of novel APIs and software tools related to the hybrid modelling of time-variant heterogeneous objects.
5. The introduction of new extensions to space-time blending within the ICs.
6. The introduction of novel approaches based on an animated “stand-ins” technique allowing us to overcome a set of known issues.
7. The retrieval and analysis of new experimental results using dynamic hybrid models.

2 Related work

In the Introduction we have mentioned a number of the existing representations and computer animation techniques. In this chapter we provide a more detailed overview of these topics. We discuss the advantages and disadvantages of the examined representations along with their main application areas. We then outline the main features of the IC framework that can be used to combine models defined in different representations and we show that the IC framework can be used as a powerful platform for the definition of hybrid models of heterogeneous objects.

In the second part of the chapter we provide a survey of the existing computer animation techniques and methods used for the definition of time-dependent models. We analyse the strengths and weaknesses of the existing approaches for the definition of dynamic models. Finally, we discuss the necessity for the extensions to the IC framework so that it can be used for the modelling of time-variant heterogeneous objects.

2.1 The boundary representation

A boundary representation (BRep) model is used to represent the shape of an object as a set of connected surface elements (i.e., the surface boundary of the object). BRep models consist of both geometric and topological information. The geometric information consists of the description of the points belonging to the surface of an object, while the topological information specifies the connections between points on its surface and allows us to specify the set of components making up the shape of the object being represented. There are a number of different types of BReps available for the definition of a model.

2.1.1 The parametric representation

Historically the parametric representation (PRep) proved to be useful for the approximation of many natural and man-made objects. Parametric representations allow us to get a straightforward mapping from parametric space to another (possibly higher dimensional) space :

$$F(P) : E^M \rightarrow E^N; P \in [a; b] \subset E^M$$

The majority of existing mapping functions used in computer graphics are variations of polynomial functions. A generalised form of a 1D polynomial function is defined as follows:

$$F(t) = \sum_{i=0}^n b_{i,n}(t) \cdot P_i$$

The shape of curve $F(t)$ is determined by a set of control points P_i . A set of intermediate points between the specified control points is computed using the provided polynomial basis function $b_{i,n}$ (see fig. 1).

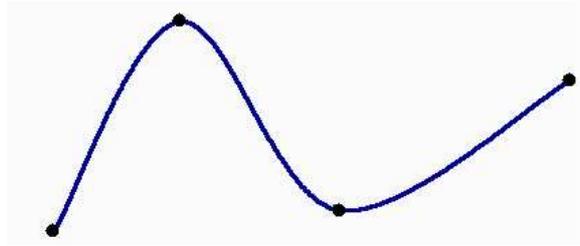


Figure 1: An example of polynomial curve passing through a set of control points.

The explicit relations between the evaluated values and a set of parameters defined through the mapping rules greatly simplify the process of model evaluation. This can be achieved through the application of a mapping rule that increases the dimensionality of the resulting object and through the combination of the parametric objects.

For instance, the combination of parametric curves can be used for the definition of surface patches (tensor products). Each of the curves has associated with it a parameter allowing us to perform a mapping from a 2D parametric space to a 3D space on the surface of the modelled object (see fig. 2). A patch of the surface of the object can be defined by a set of control points and an interpolation rule for a set of intermediate points on the surface. A set of patches of the modelled surface can be defined independently meeting certain continuity constraints at the boundaries of a patch in order to align it with its neighbouring patches on the surface.

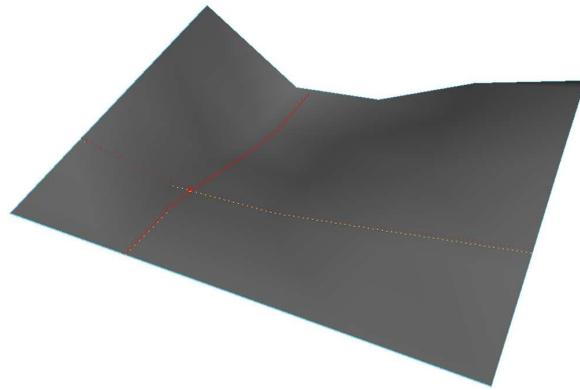


Figure 2: *A patch of a parametric surface.*

There exists a wide range of mapping functions suitable for the definition of such a model. These provide various degrees of control and can be described using different types of user-defined constraints. The main difference between the wide variety of parametric surfaces is due to the use of diverse polynomial basis functions. Some of the popular types of parametric surfaces are Bezier (Bezier, 1986), Coons (Coons, 1967) and NURBS (Versprille, 1975) surfaces. The parametric representation also provides a way to define complex deformations of surface data (Barr and Alan, 1984; Sederberg and Parry, 1986).

The parametric representation has a large number of applications in the CAD industry (Farin, 2002), including but not limited to the aerospace engineering, in the design of vehicles and in product design. Unfortunately, this representation has a number of limitations. The topology of the modelled object cannot be easily modified and needs to be considered at an early stages of the modelling process. The combination of PRep objects in one model is rather limited. There is no easy way to apply set-theoretic operations to PReps. Another serious issue is the complexity of the procedure required for the point membership classification and for the evaluation of the distance from a point to a PRep object. The latter problem is common to the majority of BReps, which significantly limits their application in solid modelling.

In classic modelling, parametric surfaces are a natural way to represent a modelled object. In the early modelling systems, a model could be rendered as a set of discrete curves on the surfaces of the objects. With advances in computer hardware, polygon rendering of a discretized patch became a more

common approach. Usually the resulting surface patches are converted to a set of planar polygons for the purposes of rendering.

2.1.2 The polygonal representation

The most widespread type of BRep is the polygonal representation. A polygonal model (G) consists of a set of vertices (V), a set of edges (E) and a set of faces (F) (Foley *et al.*, 1995):

$$G = \{V, E, F\}$$

$$V = \{v_1, \dots, v_n\}$$

$$E = \{e_1, \dots, e_l\}; e_i = \{v_j, v_k\}; 0 < j, k \leq n$$

$$F = \{f_1, \dots, f_m\}; f_i = \{e_j, e_k, \dots, e_p\}; 0 < j, k, p \leq l$$

A face usually consists of 3 or 4 edges (i.e.: $m=3,4$).

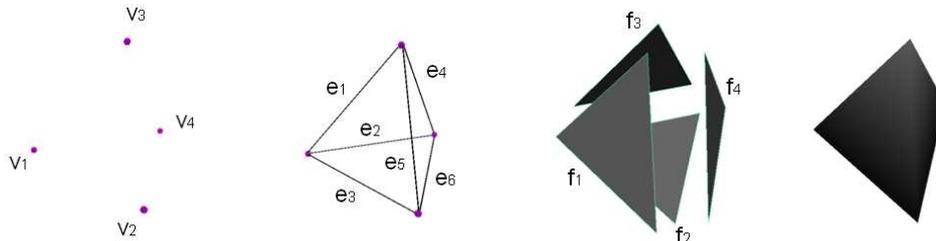


Figure 3: *The components of a polygonal model.*

Such representations of 3D models play a significant part in the world of modern computer graphics. They are widely used due to the fact that individual polygons, such as triangles and quads, provide a reasonable approximation of various surfaces that we come across in daily life. Unfortunately BRep models are not well suited for certain types of operations (such as set-theoretic and relational operations) and often require additional constraints if the topology of an object needs to be changed subsequently. It is also important to note that without additional custom data BRep models do not provide a history of the model building process. They only provide a description of the resulting object model. This shortcoming may complicate iterative work

on the model, as well as preventing an understanding of how the model was produced.

Significant research has already been conducted on fast and realistic techniques for rendering BRep models. When using a BRep representation, a high degree of realism is usually achieved by a significant increase of the number of polygons constituting the model and by increasing the complexity and sophistication of the algorithms involved in the rendering process. Polygonal BRep models are used in the majority of interactive applications and games, as well as in the production of high quality computer graphics. Despite their ability to result in realistic renderings, BRep models are only able to approximate an object's surface but are incapable of representing the internal structure of the objects being modelled. This means that the modelled objects are in effect hollow, so there is no easy way to represent or to explore their internal structure. To overcome this limitation of boundary representation models some CAD systems allow the user to represent the interior of an object by a homogeneous material. The addition of the limited homogeneous material however lessens the usefulness of boundary representations in a significant number of areas where information about the model's surface alone is not sufficient. Another serious problem of BReps is the possible introduction of errors of different nature over the course of the modelling process. Thus the resulting model may have topological and geometric defects, which requires additional post-processing in order to ensure the validity of the model (Shen *et al.*, 2001).

Even though BReps have a number of serious shortcomings, they are currently one of the most widely used representations. Various modelling and animation techniques were developed for BRep modelling over the years. There exists a substantial set of BRep modelling tools. These tools allow people of diverse skills and abilities to produce BRep models of varying quality and complexity. Vast groups of experts in the areas of static or dynamic BRep modelling do not have enough experience to work efficiently with other representations. Thus, it is important to support BRep models in a hybrid modelling system in order to make it accessible to a wider audience. Certain objects or their parts can be adequately defined using BReps in a mixed-dimensional hybrid model. Besides, the support of Boundary Rep-

resentation models allows us to exploit the significant number of the models that have been produced over four decades. Consequently it becomes apparent that BReps can play a significant part in the definition of heterogeneous objects and that BReps should be integrated into a generic hybrid modelling framework. This will allow us to take advantage of their best characteristics and to overcome some of their shortcomings when used in conjunction with other representations as the basis of a hybrid modelling framework.

2.2 The representations for volumetric models

Unlike BRep models volumetric models provide the description of the interior of an object along with its surface. These types of model are better suited for the modelling of real-life objects of a diverse nature, as objects defined with a volumetric model are actually solid and contain important information regarding the object's properties. As we shall see, volumetric models are essential for the description of real heterogeneous objects.

2.2.1 The voxel representation

One of the natural ways to represent a solid object is to store a set of samples, called “voxels” (volume elements), of the volume of space occupied by the object (Kaufman *et al.*, 1993). Each discrete sample can store a particular property of the object at a given location:

$$O = \bigcup_i^N S_i = \bigcup_{i_1}^{N_1} \dots \bigcup_{i_M}^{N_M} S_{i_1, \dots, i_M}; S_i, S_{i_1, \dots, i_M} \in \mathbf{S}$$

where O is a voxel object, S_i is the i -th sample and S_{i_1, \dots, i_M} is a sample of a voxel data in M -dimensional space. In the simplest case each sample stores a binary value of “0” (indicating that the sample is outside the object) or “1” (indicating that the sample is inside or on the surface of the object). In the general case samples can store arbitrary information. 3D scenes represented in such a way can be captured from real life using different voxelisation techniques (such as computer tomography (CT), magnetic resonance

imaging (MRI), ultrasonography, geophysical measurements, etc. (Nielson, 1999)) or be modelled on a computer (Bærentzen, 2002; Bremer *et al.*, 2002; 3D Coat, 2010) allowing us to integrate models of a diverse nature. Models of this type are able to store significantly more detailed information about the represented objects and their relations, allowing the end-user to explore and manipulate the objects (i.e. to analyse the objects' interior structure or to access properties assigned to various parts of the object's interior) without the familiar restrictions found in polygonal models.

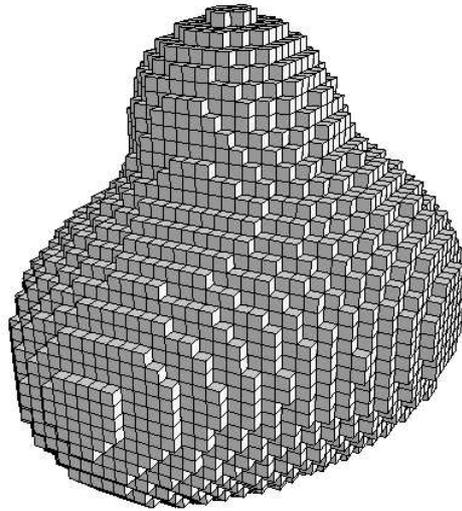


Figure 4: *Voxel data.*

Voxel models support a number of important operations helping us to create new models and to modify existing models (Kaufman *et al.*, 1993). Note that the voxel representation is resolution dependent. It is thus desirable to create a model of the highest possible resolution to avoid major aliasing artefacts. Various interpolation schemes are used for the interpolation of the discrete voxel samples in the volume. One has a choice of a number of interpolating functions (Barthe *et al.*, 2002). A specific interpolation scheme can be selected depending on the desired result (Frisken, 1999; Kadosh *et al.*, 2003):

$$F(P) = \sum_{i=N_x^s}^{N_x^e} \sum_{j=N_y^s}^{N_y^e} \sum_{k=N_z^s}^{N_z^e} G_{ijk}(P) \cdot S_{ijk}; P \in E^3$$

where $F(P)$ is the interpolated value of the volumetric function, $G_{ijk}(P)$ is

the interpolation function, S_{ijk} is the sample value of the element $\{i, j, k\}$ and $N_x^e - N_x^s$ defines the number of samples required for the evaluation of volumetric value at point P .

Another major problem with voxel models is the necessity to store information about the entire volume set. For instance, a grid of 512 x 512 x 512 units of volume requires the storage of approximately 134 million voxels. There are some well-known techniques allowing us to compress this information (Jones *et al.*, 2006; Komma *et al.*, 2007; Forstmann *et al.*, 2007). Most of these techniques are suited for static data and cannot easily be applied to dynamically changing models. One possible solution is the CROVE system proposed in (Chen *et al.*, 1999). This system allows us to organise hierarchical structures and store the voxel data set only on a per object basis (i.e. the scene consists of objects each of which contains its own local voxel data set). In the past, a number of computer games utilised a limited subset of the voxel representation (Moby Games™, 2010). Thus, some of these games could provide the user with more freedom in the interaction with the environment and even let the user modify the game world.

A more advanced voxel representation provides us with additional information regarding the modelled object. Instead of storing a simple binary enumeration value, we could store a distance to object's surface (see fig. 5). Uniform distance information can be retrieved from binary voxel data through the application of the Distance Transform (DT) (Frisken, 1999). The interpolation of distance values inside the volume could be retrieved using one of the interpolation schemes (Barthe *et al.*, 2002). Due to the discrete nature of voxel data, a signed distance can only be defined within limited area of space. For an arbitrary point in space a special blending of the "near field" (within the bounds of the area providing the sample of distance values) and the "far field" (outside the bounds) is required (Sigg, 2006):

$$F(P) = w \cdot F_{near}(P) + (1 - w) \cdot F_{far}(P); w \in [0; 1]; P \in E^N$$

The signed distance information could be useful for the application of complex shape modelling operations (Schroeder *et al.*, 1994; Adzhiev *et al.*, 2000), physical simulation (Jones, 2003) and a number of other applications (Jones

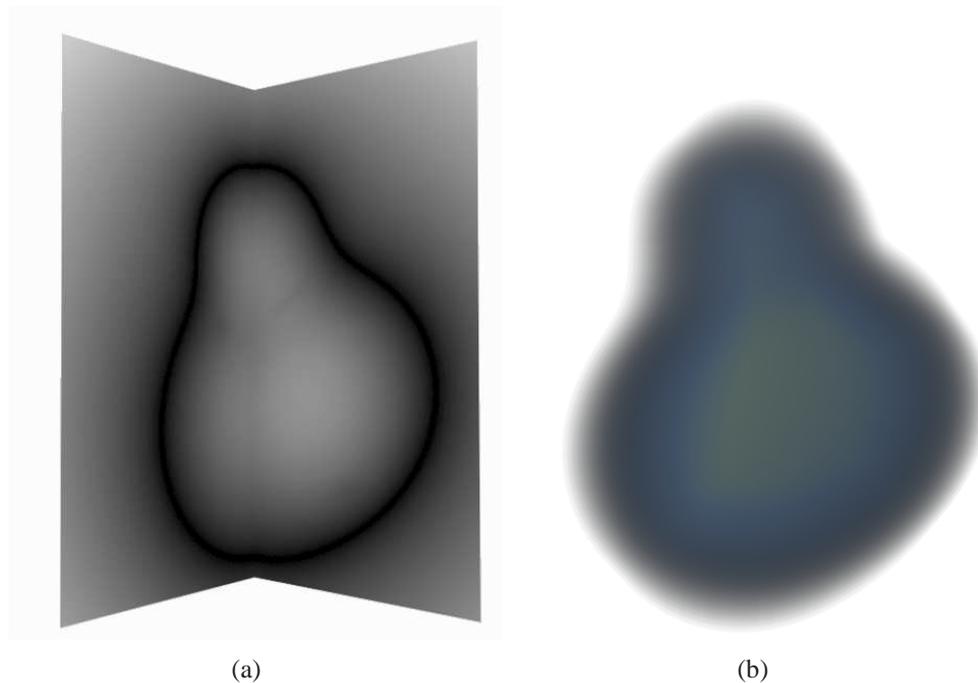


Figure 5: *Volumetric data stored as discretized distance field (a) Two orthogonal slices of distance data (b) Volume rendering of volumetric data with colour variation.*

et al., 2006). This voxel representation has even higher storage requirements, because instead of binary values, scalar or even vector distance values (Kobbelt *et al.*, 2001) need to be stored. There exists a number of ways to compress this data. For instance, Frisken proposed the usage of Adaptively Sampled Distance Fields (ADFs) (Frisken *et al.*, 2000) and Bærentzen proposed a similar method Bærentzen (2002), relying on the hierarchical adaptive storage of signed distance fields (see fig. 6). But these methods do not guarantee lossless compression and generally introduce additional approximation errors to the model. These errors result in the introduction of discontinuities, which may cause unacceptable artefacts during subsequent modelling stages.

The voxel representation is still a good choice for the description of natural objects. The volumetric data from real heterogeneous objects can be captured using existing 3D scanning technologies commonly used in medicine and the manufacturing industries. It is frequently important to be able to work with real models. Models of man-made or natural objects should be able to be

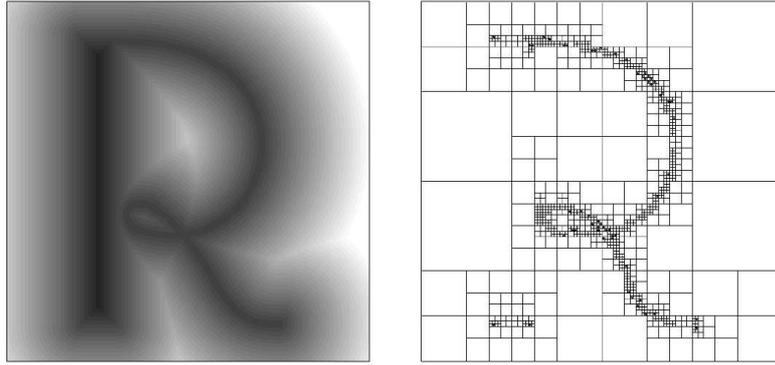


Figure 6: An example of adaptively sampled distance field in 2D and the underlying adaptive data structure (Friskin et al., 2000).

easily incorporated into a hybrid model in order to produce complex heterogeneous objects. Unlike BReps, voxels allow us to store information relating to the object’s internal structure and certain physical properties. This allows us to represent real heterogeneous objects at a predefined resolution. Thus, voxel data retrieved from a specific field can be used in a hybrid model without the necessity for the user to provide its description manually. Hence, the proposed Dynamic IC framework needs to support the voxel representation.

2.2.2 Implicit surfaces

Another way to represent an object in n -dimensional space is to define a set of points satisfying a specific condition:

$$S = \{(x_1, \dots, x_n) : F(x_1, \dots, x_n) == true\}$$

where $F(x_1, \dots, x_n)$ is a predicate function returning a value indicating whether a given point (x_1, \dots, x_n) belongs to this set. A scalar function defined in n -dimensional Euclidean space together with an inequality function can be used to define an object (Bloomenthal, 1997):

$$P \in R^N$$

$$F(P) = f(P) > T; T \in R$$

$$f(P) : R^N \rightarrow R$$

Any point $P \in R^N$ in this space can be classified according to following conditions:

$$\begin{cases} f(P) > T, P \text{ is inside the object} \\ f(P) = T, P \text{ is on the object's boundary} \\ f(P) < T, P \text{ is outside the object} \end{cases} \quad (1)$$

where T is a threshold value or an iso-level.

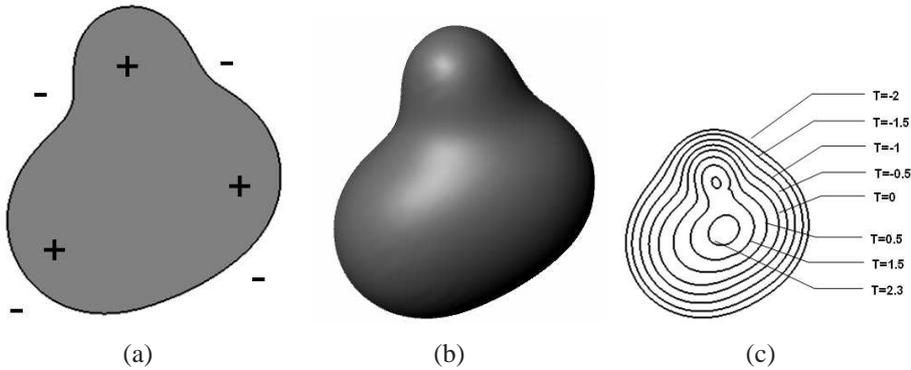


Figure 7: A scalar field (defining function): (a) The sign of a scalar field, (b) The extracted implicit surface ($T=0$) and (c) Different iso-surfaces for different values of T .

In geometric modelling $f(P)$ is usually defined in 3D Euclidean space. The subset $\{P \in R^N : f(P) \geq T\}$ is called a solid object and the subset $\{P \in R^N : f(P) = T\}$ is called an iso-surface.

It might be easier to understand how a scalar function is defined if we refer to the previous section. A voxel representation allows us to store discrete samples of the volume contained in an object. In the simplest case we know that a sample is inside the object if the voxel stores a non-zero positive value. The same approach can be extended for any point in space instead of only discrete voxels. We can define a scalar field that uses point coordinates as input parameters and returns a scalar value. The returned scalar value can be interpreted in the same way as the value stored in a voxel. Finally, if the scalar value returned for the point provided to the defining function as input is greater than a threshold value, this point is located inside the object. If the

returned scalar value is equal to the threshold value, the point is located on the surface of the object and finally if the scalar value is less than the threshold value, the point is located outside the object. The scalar value returned by the function can be interpreted as a distance to the surface of the object described by the function. But this does not have to be a distance in a Euclidean sense. For example, a defining function for a disk of radius two centred at the origin of coordinate axes would look like:

$$f(x, y) = 2^2 - x^2 - y^2$$

In this particular case, we are interested in a disk with a threshold value of zero, i.e. we are looking for values greater than or equal to $T = 0$.

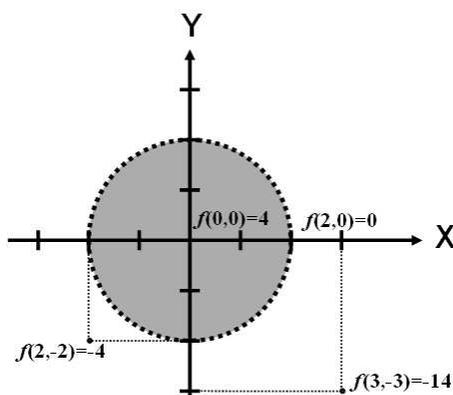


Figure 8: An example of an implicit object.

The grey area in figure 8 is a subset of the 2D plane where the values returned by the function $f(x, y)$ are positive (for instance, $f(0, 0) = 4$). The dashed contour is a set of points where the function $f(x, y)$ returns the value 0 (for example, point $(2, 0)$ lies on the perimeter of a circle and $f(2, 0) = 0$). For any point of the 2D plane outside the dotted circle $f(x, y)$ returns negative values (e.g. $f(3, -3) = -14$). Hence any point on the 2D plane can be classified. The set of points for which the function $f(x, y)$ returns values greater than or equal to 0 describes the set of points that belong to the object being described. In this fashion we can describe arbitrarily complex objects in n -dimensional space using arbitrarily complex functions.

Implicit representations have been successfully employed in many dif-

ferent application areas, such as collision detection (Savchenko and Pasko, 1995), precise contact modelling (Desbrun and Gascuel, 1995), sketch modelling (Tai *et al.*, 2004), the design of natural shapes (McCormack and Shershtyuk, 1998), virtual surgery (France *et al.*, 2005), the modelling of anatomical structures (Oeltze and Preim, 2004), 3D point cloud approximation (Ohtake *et al.*, 2004), the simulation of natural phenomena (Stora *et al.*, 1999), metamorphosis animations (Barbier *et al.*, 2005), image recognition (Tong *et al.*, 2005) and in path-planning (Nguyen, 2007) among others.

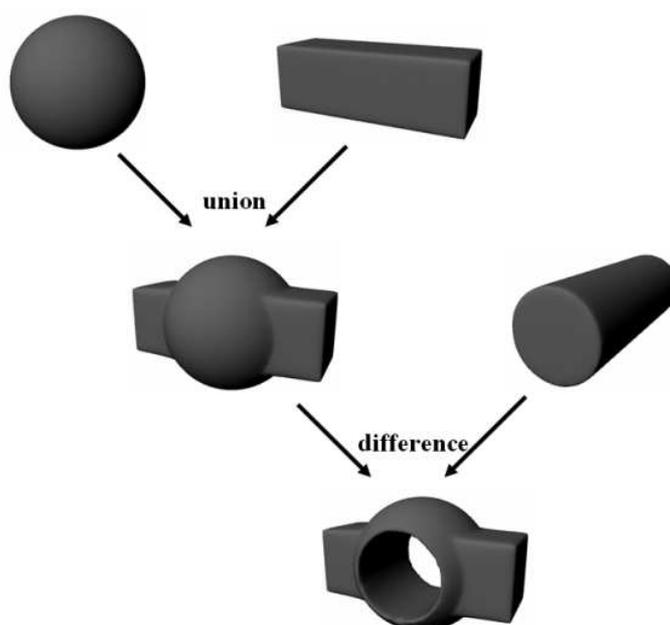


Figure 9: An example of a CSG tree.

In Constructive Solid Geometry (CSG) it is possible to construct complex objects using set-theoretic operations on a set of implicit objects. The assembled model is represented as a tree (fig. 9). Primitives or objects are stored in the leaf nodes of the CSG-tree, while operations used to combine the primitives are stored in the interior nodes of the tree. A tree can be represented as a defining function $f(P) : R^N \rightarrow R$. In which case, the subset $\{P \in R^N : f(P) \geq T\}$ defines the geometry of the resulting object. Thus, one needs to traverse the CSG-tree for a set of points in order to evaluate the geometry of the final model. Unlike BReps implicit representations make it possible to store information about a solid model. Though usually only binary point-membership classification information is provided (i.e. the

CSG-model can only be used to find out whether particular point in space is inside or outside the object). CSG representations are widely used by the CAD-community, partly because they represent a natural and simple way to construct solid objects. Also in some cases CSG-trees can be used as a guide to the manufacturing process.

Wyvill et al. (Wyvill *et al.*, 1999) incorporated the CSG approach with skeletal-based implicit surfaces. They called this representation a “Blob-Tree”. The “BlobTree” was used for the texturing of implicit surfaces (Tigges and Wyvill, 1998), the creation of an accurate biological model of the sea shell (Galbraith *et al.*, 2000), the modelling of smoothly branched trees (Gal), the controlled metamorphosis (Galín *et al.*, 2000; Barbier *et al.*, 2005) and the sketch-based modelling (Schmidt *et al.*, 2005).

We can see that over the years implicit surfaces have found a large number of applications in different areas of shape modelling and animation. Simple point classification rules can be useful for the definition of complex volumetric structures needed for the definition of space partitions occupied by heterogeneous objects or their parts. Smooth shapes are often produced using implicit surfaces and have proven to be especially useful for modelling of natural organic models. As we have already outlined many of the existing natural models are heterogeneous, and their geometry could be adequately captured using implicit surfaces. The internal structure and the geometric shape of natural objects can undergo significant modifications over time. It is desirable to be able to reflect this process in our dynamic model. A known feature of implicit surfaces (particularly important for time-variant heterogeneous object modelling) is the ease of modelling of objects with changing topology. This provides the user with additional flexibility for the definition of dynamic objects without serious constraints of their topological properties. Integration of implicit surfaces into our hybrid modelling framework provides us with another degree of freedom in the definition of complex time-dependent heterogeneous objects.

2.2.3 The Function Representation (FRep)

An FRep is a generalised model representation allowing us to define objects of an arbitrary dimensionality and to mix objects of different dimensionalities within a single model. An FRep may incorporate a number of implicit surfaces, CSG models, voxel representations (Adzhiev *et al.*, 2000) and a rich set of operations (Pasko *et al.*, 1995). We can already see that FReps are a powerful representation with some of the advantages of other representations suitable for heterogeneous object modelling. An FRep system comprises a set of geometric primitives, operations and relations. Primitives are combined together using operations. Relational operations can be used to classify subsets within the model. An FRep usually allows us to represent advanced models compactly and in a resolution independent manner. A constructive tree used with an FRep naturally simplifies iterative work on a model. It makes it easier to see how the model was assembled and how to modify the model if required. This is vital not only for static models but also for dynamic models, because it gives us the freedom to introduce dramatic changes to both the geometry and the topology of the model. It is also important to note that an FRep can be used to specify arbitrary volumetric attributes of the modelled objects so as to represent their internal structure in different contexts (Schmitt *et al.*, 2001). Hence even complex heterogeneous objects can be modelled using this kind of representation.

The FRep model

FRep models can be specified in n -dimensional space, but the main modelling domain is considered to be the 3D Euclidean space. An FRep can be described as an algebraic system (Pasko *et al.*, 1995):

$$(M, \Phi, W)$$

where M is a set of geometric objects, Φ is a set of geometric operations, W is a set of relations specified for the set of objects M .

FRep objects

An FRep geometric object is defined as a closed subset of R^N in the fol-

lowing way:

$$G = \{(x_1, \dots, x_n) : F(x_1, \dots, x_n) == true\}$$

$$F(P) = f(P) > T; T \in R; P \in R^N$$

$$f(P) : R^N \rightarrow R$$

where $f(P)$ is called the defining function. FRep objects can be divided into two groups: primitives and complex objects (Pasko and Adzhiev, 2002). Primitives are defined by a concrete function chosen from a predefined set or created by the user. Any primitive can be interpreted as a black box, defined by a function that can be evaluated in the modelling space. In the general case, a complex object can be obtained by a combination of operations (as described in section 3.6) applied to a set of primitives. A resulting complex object can also be represented in the form of a defining function. Such objects are defined within a model and are usually created using a CSG approach.

FRep operations

A set of FRep geometric operations consists of n -ary operations closed on the object representation (Pasko and Adzhiev, 2002):

$$\Phi_i : M^1 + M^2 + M^n \rightarrow M$$

where n is the number of operands of the operation. It should be apparent that the result of an operation is an object defined on the object set M , thus guaranteeing the closure property of FReps. A general n -ary operation is defined in the following manner:

$$G_{n+1} = \Phi^n(G_1, \dots, G_n); G_1, \dots, G_n, G_{n+1} \in M$$

$$f_{n+1}(P) = \Psi(f_1(P), \dots, f_n(P)); \Psi(R, \dots, R) : R \times \dots \times R \rightarrow R$$

$$f(P) : R^N \rightarrow R$$

where Ψ is a continuous real function of n variables and $f_i(P)$ are defining

functions. An example a binary operation would be defined as follows:

$$G_3 = \Phi^2(G_1, G_2); G_1, G_2, G_3 \in M$$

$$f_3(P) = \Psi(f_1(P), f_2(P)); \Psi(R, R) : R \times R \rightarrow R$$

$$f_1(P), f_2(P), f_3(P) : R^N \rightarrow R; P \in R^N$$

Functions used in an FRep are required to be at least C^0 -continuous. But it is preferable to use at least C^1 -continuous functions. The first derivative of the function can be used to compute the gradient and the normal on the object's surface:

$$\nabla f(P) = \left(\frac{\partial f(P)}{\partial x_1}, \dots, \frac{\partial f(P)}{\partial x_N} \right); \nabla f, P \in R^N$$

$$\vec{n}(P) = -\frac{\nabla f(P)}{\|\nabla f(P)\|}; \vec{n}(P), P \in R^N$$

The C^1 -discontinuity of the defining functions can lead to rendering artefacts and ambiguities of the model if further operations are applied (Fayolle, 2006). In complex models the C^1 -discontinuity of the defining function can even lead to the introduction of C^0 -discontinuity of the objects, if the gradient is used for their defining function. In the original work on CSG modelling (Ricci, 1973) the following functions were proposed for set-theoretic operations:

$$S_1 \cup S_2 = \max(d_1(P), d_2(P))$$

$$S_1 \cap S_2 = \min(d_1(P), d_2(P))$$

where S_i is a solid and $d_i(P)$ is a distance function to the boundary of the solid. The application of min/max functions to analytical functions results in a C^1 -discontinuous function. FRep modelling provides at least C^1 -continuous set-theoretic operations based on the theory of R-functions:

$$S_1 \cup S_2 = f_1(P) + f_2(P) + \sqrt{f_1^2(P) + f_2^2(P)}$$

$$S_1 \cap S_2 = f_1(P) + f_2(P) - \sqrt{f_1^2(P) + f_2^2(P)}$$

where $f_i(P)$ is the defining function of the object. Both defining functions $f_1(P)$ and $f_2(P)$ are expected to exhibit distance properties. Figure 10 shows

the plots of two set-theoretic operations. The values of f_1 are placed along the x -axis, while these of f_2 are placed along the y -axis and the height of the surface at location (x, y) shows/represents the value returned by the set-theoretic operation $F(f_1(P), f_2(P)) = F(x, y)$. It should be apparent that the union operation returns a positive value whenever either of its operands are positive (i.e. the resulting object consists of the space that lies in the interior of either input object), while the intersection operation is positive only when both of its arguments are positive (i.e. the resulting object consists of the space that lies in the interior of both input objects).

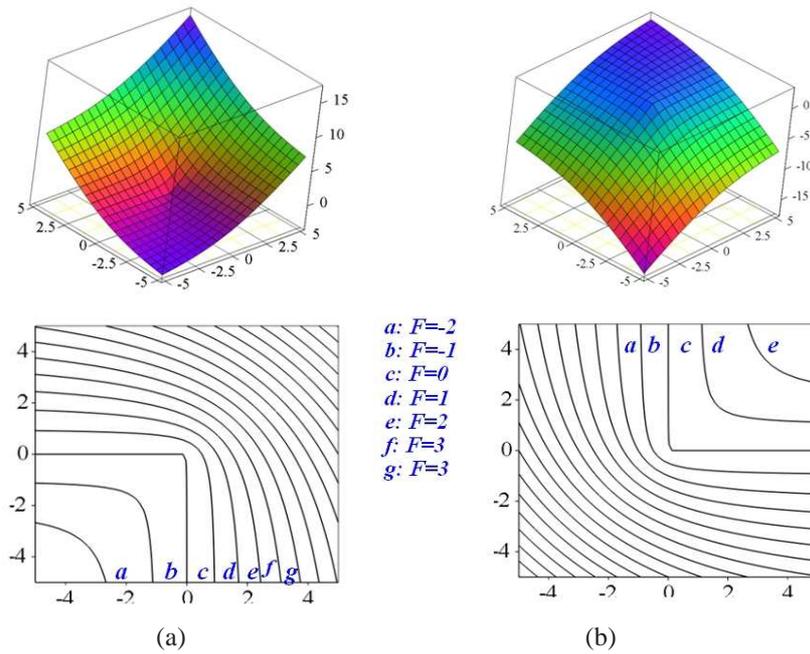


Figure 10: Set-theoretic operations based on R-functions: (a) Union and (b) Intersection.

If a particular type of continuity is required, the CSG operations can be generalised in the following form (Pasko and Savchenko, 1994):

$$S_1 \cup S_2 = f_1(P) + f_2(P) + \sqrt[k]{f_1^{\frac{1}{k}}(P) + f_2^{\frac{1}{k}}(P)}$$

$$S_1 \cap S_2 = f_1(P) + f_2(P) - \sqrt[k]{f_1^{\frac{1}{k}}(P) + f_2^{\frac{1}{k}}(P)}$$

which guarantees C^k -continuity.

In addition to the traditional CSG operations FReps provide a more flexible way for the creation of complex models by blending set-theoretic operations.

These operations are also based on R-functions:

$$F_b(f_1, f_2) = f_1 + f_2 + \sqrt{f_1^2 + f_2^2} + disp_b(f_1, f_2)$$

$$disp_b(f_1, f_2) = \frac{a_0}{1 + \left(\frac{f_1}{a_1}\right)^2 + \left(\frac{f_2}{a_2}\right)^2}$$

where a_1 controls the contribution of the first function, a_2 controls the contribution of the second function and a_0 controls the overall “shift” of the resulting function. Set-theoretic blending operations allow us to dramatically change the resulting shape by controlling the influence of each of the initial shapes being blended, as well as by controlling the overall offset from the resulting shape.

Another important operation available in FReps is that of bounded blending operation (Pasko *et al.*, 2005):

$$F_{bb}(f_1, f_2, f_3) = f_1 + f_2 + \sqrt{f_1^2 + f_2^2} + disp_{bb}(r(f_1, f_2, f_3))$$

$$disp_{bb}(r(f_1, f_2, f_3)) = \begin{cases} \frac{(1 - r^2(f_1, f_2, f_3))^3}{1 + r^2(f_1, f_2, f_3)}, & r(f_1, f_2, f_3) < 1 \\ 0, & r^2(f_1, f_2, f_3) \geq 1 \end{cases}$$

$$r(f_1, f_2, f_3) = \begin{cases} \frac{r_1^2(f_1, f_2)}{r_1^2(f_1, f_2) + r_2^2(f_3)}, & r_2 > 0 \\ 1, & r_2(f_3) = 0 \end{cases}$$

$$r_1^2(f_1, f_2) = \left(\frac{f_1}{a_1}\right)^2 + \left(\frac{f_2}{a_2}\right)^2$$

$$r_2^2(f_3) = \begin{cases} \left(\frac{f_3}{a_3}\right)^2, & f_3 > 0 \\ 0, & f_3 \leq 0 \end{cases}$$

where f_1 and f_2 are the defining functions of the two objects being blended, f_3 is a defining function of the bounding solid, specifying the subset of space where blending takes place; a_1, a_2 allow the user to control the blend sym-

metry and a_3 defines the influence of the bounding solid on the shape of the resultant blended object. Other operations available in FReps are affine transformations, basic deformations (Barr and Alan, 1984), non-linear deformations (Schmitt, 2002), metamorphosis and many others.

Unfortunately the use of R-functions changes the distance properties of the scalar field produced by the defining function. If this is an undesirable effect, an additional set of operations based on the Signed Approximate Distance Functions (SARDF) can be employed (Fayolle *et al.*, 2006). An SARDF provides a good and controllable approximation of the exact distance function, which is also C^1 -continuous. All CSG and blending operations can be redefined using SARDFs.

FRep relations

Relations in FReps are subsets of the set M^N , which is a Cartesian product of the sets M . They are generally defined as n -valued predicates:

$$S : M \times \dots \times M \rightarrow I^K$$

where $I^K = i_1, \dots, i_K$ is considered to be a set of integer values in a K-ary logic. Relational operations can be used to classify subsets within the FRep model. For example, the point membership relation allows us to find out whether an arbitrary point in space belongs to an FRep object:

$$G_1 = \{P : f_1(P) \geq T\}; f_1(P) : R^N \rightarrow R$$

$$S_3(P, G) : \begin{cases} 0, f_1(P) < T; (P \notin G_1) \\ 1, f_1(P) = T; (P \in \partial G_1) \\ 2, f_1(P) > T; (P \in G_1) \end{cases} \quad (2)$$

Another important relation defined in FReps is the intersection relation. This is defined by the two-valued predicate that can be used to determine if two FRep objects intersect (i.e. if they have common points in the modelling space). Such a relation can be employed for collision detection between FRep objects. In practice one has to apply numerical methods performing discrete sampling within the bounding volume of the objects in order to detect collisions between complex objects (Savchenko and Pasko, 1995).

Hypervolume extensions for FReps

It is often desirable to be able to represent specific properties of the geometric model. In the simplest case one might want to associate a constant colour or a texture with a geometric object. A wide range of texturing techniques was developed in the past (Heckbert, 1986). Not all of the existing texturing methods are applicable to volumetric objects. One of the approaches suitable for arbitrary models requires the specification of a solid procedural texture defined in the volume (Perlin, 1985). The definition of a global complex solid texture is neither intuitive nor flexible enough. Because commonly such a texture can be created and modified only by an experienced user. Another way to enhance a purely geometric model is to associate optical attributes with the object's geometry (Chen *et al.*, 1999; Tigges and Wyvill, 1998). The necessity to associate the geometry of an object and its physical properties is in some cases very limiting and is mostly suitable for simple homogeneous objects. Many objects in the real world do not have an explicit relation between their geometric and physical properties. An independent representation of a point set and its attributes represents a more general approach, which allows us to have a more flexible representation of complex volumetric heterogeneous models. Such a representation also provides the user with more flexibility when specifying dynamic models.

When used in the constructive Hypervolume framework (Schmitt *et al.*, 2001), the expressive power of FReps had to be augmented in order to allow us to model more than just geometric properties. An object is defined as a tuple of its geometric set and a set of its attributes:

$$M = (G, A_1, \dots, A_k) : (f(P), S_1(P), \dots, S_k(P)); P \in R^N$$

$$f : R^N \rightarrow R$$

$$S_i : R^N \rightarrow R^M$$

where G is a point set in R^N and A_i is a multidimensional attribute defined by the function S_i , which can be interpreted as a space partition function defining specific multidimensional attributes within its interior. The defining function f has to be at least C^0 -continuous, while S_i does not necessarily have to be

continuous. Space partitions can be specified using all of the existing FRep primitives and operations.

An FRep definition of an n -ary operation is extended in the following way:

$$\Phi^n(M_1, \dots, M_n) = \Phi^n(G_1, \dots, G_n, A_{1_1}, \dots, A_{1_k}, \dots, A_{n_1}, \dots, A_{n_m})$$

$$M_{n+1} = \Phi^n(M_1, \dots, M_n); M_1, \dots, M_n, M_{n+1} \in M$$

where n is the number of operands of an operation. Such an operation requires n Hypervolume objects (each composed of a geometric object and the set of its attributes) as its input and results in a new Hypervolume object:

$$(G_{n+1}, A_{(n+1)_1}, \dots, A_{(n+1)_p}) = \begin{pmatrix} \Psi(P, f_1, S_{1_1}, \dots, S_{1_k}, \dots, f_n, \dots, S_{n_1}, \dots, S_{n_m}), \\ \Omega_1(P, f_1, S_{1_1}, \dots, S_{1_k}, \dots, f_n, \dots, S_{n_1}, \dots, S_{n_m}), \\ \vdots \\ \Omega_p(P, f_1, S_{1_1}, \dots, S_{1_k}, \dots, f_n, \dots, S_{n_1}, \dots, S_{n_m}) \end{pmatrix}$$

The resulting Hypervolume object is derived from a set of input Hypervolume objects using the transfer function Ψ to define a geometric set and a set of functions Ω which in turn define the attributes of the object. The requirements for Ψ and Ω_i are the same as for the defining function and the attribute function of the Hypervolume object. It should be apparent that in the general case attributes can affect and be affected by the geometry of the object. This allows us to establish complex relations between the various properties of the Hypervolume objects in the model. It is also important to note that Hypervolume objects involved in the operations are not required to have an equal number of attributes.

Earlier we stated that all the existing FRep entities and their compositions can be used to specify the space partitions used to define the set of attributes of a Hypervolume object. Therefore the constructive approach traditionally used in geometric modelling can be employed to specify the various properties of the modelled objects. Each attribute can have a corresponding constructive FRep tree associated with it. Thus, an attribute value can be evaluated for any given point in space. Attribute evaluation is, in a way, similar to point classification in relation to the point set enclosed by the object (see equation

(2):

$$M = (G, A_1, \dots, A_k) : (f(P), S_1(P), \dots, S_k(P)); P \in R^N$$

$$A_i : \begin{cases} S_i(P), f(P) \geq T \text{ or } g_i(P) \geq 0 \\ \theta_{1_i}, f(P) \geq T \text{ and } g_i(P) < 0 \\ \theta_{0_i}, f(P) < T \text{ and } g_i(P) < 0 \end{cases}$$

where $f(P)$ defines the *object* geometry and $g_i(P)$ defines a space partition for the i -th attribute. Two default attributes need to be introduced. θ_{1_i} which is used in cases where point P is in the interior of the geometric object, but does not belong to any space partition (i.e. a geometric object is not fully covered by the attribute space partitions). θ_{0_i} which is associated with the points outside the Hypervolume object. This default attribute usually has to conform to the requirements of the volumetric rendering system (Schmitt, 2002), since points outside the geometric set and space partitions are commonly treated as fully transparent. Components of an example constructive Hypervolume model are shown in the fig. 11.

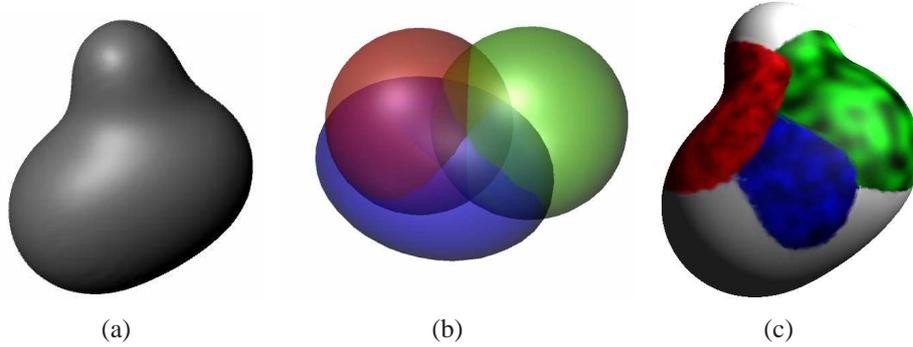


Figure 11: *The constructive Hypervolume model: (a) The original geometric object, (b) A visualisation of the space partitions and (c) The resulting Hypervolume object.*

Relations within the Hypervolume framework are defined as follows:

$$w_i(O_1, \dots, O_n, Q_1, \dots, Q_m) :$$

$$\Gamma(f_1(P), S_{1_1}(P), \dots, S_{1_k}(P), f_n(P), S_{n_1}(P), \dots, S_{n_l}(P), q_1(P), \dots, q_m(P));$$

$$P \in R^N$$

where $q_i(P)$ is a function representing a non-Hypervolume object (such as the values returned by predicates) and Γ is a predicate function.

One of the most common relations is the point membership relation (Schmitt, 2002):

$$M = (G, A_1, \dots, A_k) : (f(P), S_1(P), \dots, S_k(P)); P \in R^N$$

$$P_A = (P, A) : (P, a_1, \dots, a_k)$$

$$\Gamma(P_A, M) = \begin{cases} 0, f(P) < T \text{ or } \exists j : S_j(P) \neq a_j \\ 1, f(P) = T \text{ and } S_j(P) = a_j \\ 2, f(P) > T \text{ and } \forall j \in [1, k] : S_j(P) = a_j \end{cases}$$

A point P_A in the Hypervolume space (i.e. a point and a set of attributes) belongs to the Hypervolume object M , if it is located in the interior or on the boundary of the Hypervolume object and the attributes of the point are equal to the attributes of the object.

This set of extensions allows us to use Hypervolumes for the definition of heterogeneous objects and to describe the arbitrary set of their internal properties.

Overall FReps allow the user to design, explore and interact with models in a more natural manner, overcoming the limitations of the currently widespread boundary representation models. FRep models have been used in a wide range of applications, such as in: virtual sculpting (Schmitt *et al.*, 2004), rapid prototyping (Adzhiev *et al.*, 2005), web-based modelling (Fayolle *et al.*, 2005; Cartwright *et al.*, 2005), cultural heritage preservation (Shanat *et al.*, 2002; Vilbrandt *et al.*, 2004), hair modelling (Sourin *et al.*, 1996) and in numerical analysis (Kartasheva *et al.*, 2003) among others.

One of the significant shortcomings of FRep models is their computationally expensive model evaluation procedure. Another issue is their inadequate support in commercial packages and the absence of certain techniques used in traditional computer animation, which limits the use of FReps in a number applications. Additionally, given that there is an abundance of existing polygonal object models it is often desirable to be able to use these in conjunction with FRep models.

The Constructive Hypervolume extensions to FReps provide us with the ability to define time-dependent heterogeneous objects. Natural support of

mixed-dimensional objects within the model makes FReps suitable for the definition of multidimensional internal structures and relations between the objects in a model. FReps can naturally incorporate implicit surfaces, solid CSG models and voxels, which allows us to integrate models of a diverse nature into a single FRep model. There are methods available for a limited integration of BRep models into the Function Representation. But due to these limitations at the moment the incorporation of BReps and FReps seems more reasonable in terms of a hybrid model. It is important to note, that volumetric attributes allow us to accurately define an arbitrary number of properties of the heterogeneous objects. Unlike voxel representation, FReps are well suited for the definition of dynamic models independent of the representation used for the definition of the geometry of the object. Overall FReps are one of the most suitable and practical representations for heterogeneous object modelling. FReps have to play a significant part in the hybrid modelling framework that we are designing. Hence, a lot of attention in this thesis is paid specifically to Function Representations.

2.3 Heterogeneous objects modelling

2.3.1 Hybrid modelling

In previous sections we have outlined a number of the existing model representations. We have provided a brief description of their advantages and disadvantages. From this information it can be concluded that there is no best representation suitable for the solution of the diverse problems inherent in heterogeneous objects modelling. Each representation has its applications in various areas and allows us to resolve certain problems. This naturally led to the idea of unified hybrid models, where objects in different representations coexist within one model.

One of the early attempts to incorporate diverse representations in one model was described in (Adzhiev *et al.*, 2000). The hybrid voxel-function representation supports a two-way conversion between the FRep and voxel representations, thus allowing us to combine these two representations in one model. This can be a powerful approach for high-precision modifications

and modelling of acquired real-world volumetric objects. Unfortunately, this work did not describe how other model representations could be integrated into the proposed framework.

Allègre *et al.* introduced the concept of the HybridTree (Allègre *et al.*, 2004), where skeletal implicit surfaces and BRep meshes could be combined in an extended CSG-tree. This allowed the inventors of the HybridTree to take advantage of CSG, certain types of implicit surfaces and polygonal meshes. In this fashion modelling of complex shapes could be performed through the combination of objects of a diverse nature (Allègre *et al.*, 2006). Another interesting application proposed by the authors is the partial restoration of polygonal meshes. Similar to (Allègre *et al.*, 2004) Fougerolle *et al.* introduced a hybrid constructive tree (Fougerolle *et al.*, 2005), which had leaves with both implicit and parametric representations (the provided examples demonstrated usage of supershapes (Gielis *et al.*, 2003)), while the nodes of the tree are CSG operations based on R-functions.

Although both of the above methods support hybrid modelling, they are mostly centred on the surfaces of the modelled objects. A structured volumetric layered model composed of different materials (Cutler *et al.*, 2002) is designed to work with solid objects. The authors used polygonal objects, volumetric objects and implicit surfaces as input to their system. All these models were converted to a tetrahedral mesh that could be used for further editing and simulation using FEM. This approach provided limited information about the topology of the model and did not provide sufficient tools for the creation of complex dynamic heterogeneous objects. The proposed framework was tailored for specific modelling and simulation tasks, but it could be further extended to be able to solve a larger set of problems.

In this research project we are concerned with the modelling of volumetric heterogeneous objects. It is also important to note, that a hybrid model of a heterogeneous object can be constructed from objects of different dimensionalities, which the aforementioned approaches do not allow us to do. These multidimensional objects can then be linked through different types of relations, allowing us to understand how these objects are bound to each other. Rossignac and O'Connor (Rossignac and O'Connor, 1990) observed that het-

erogeneous objects may include components of different dimensionalities and they employed the notion of a Topological Complex combining mixed-dimensional cells to accommodate such components (Rossignac, 1997). A working group of the British Geometric Modelling society developed a special purpose application programming interface (API) for Solid Modelling, called Djinn (Armstrong *et al.*, 2000) based on objects partitioned in a cellular fashion and containing cells of differing dimensionalities.

Kumar *et al.* (Kumar *et al.*, 1999) proposed the usage of the so-called fiber bundles, which allow them to independently represent the geometry and an arbitrary set of attributes of the heterogeneous objects. Additional mapping functions associated with the attributes perform the mapping from the geometry model to the attribute model. Biswas *et al.* presented a distance field based approach for modelling heterogeneous objects in which space is parameterised using the distance of a given point from the geometric boundaries of the object (Biswas *et al.*, 2004). A thorough overview including the requirements and comparisons of the existing heterogeneous objects modelling systems is provided in (Kou and Tan, 2007).

Past research in the area of heterogeneous objects modelling was mostly centred around static models. Dynamic hybrid models are still an on-going research topic (Xiaoping and Dutta, 2003; Adzhiev *et al.*, 2002) with a large set of unanswered questions. Kou and Tan (Kou and Tan, 2007) notice that dynamic heterogeneity modelling can have a wide range of applications in such areas as biomedicine, surgery simulation, mechanical engineering and others. We will present a more detailed discussion regarding this topic in section 2.4.

We can see from the cited works that heterogeneous object modelling is an important on-going research area. Naturally, heterogeneous objects modelling was initially primarily focused on static objects. In this thesis we mainly focus on the problems of time-dependent heterogeneous object modelling. In order to achieve this, we rely on valid existing frameworks for static heterogeneous object modelling. Modelling of time-variant heterogeneous objects was not thoroughly researched in the past and still remains a challenging task.

2.3.2 Implicit Complexes (IC)

One of the available frameworks allowing us to perform modelling of heterogeneous objects is the Implicit Complexes Framework. In 2002, Implicit Complexes (ICs) (Adzhiev *et al.*, 2002) were introduced as a representation for a cellular-functional hybrid model of heterogeneous objects. The hybrid model described within ICs may contain entities of diverse dimensions and representations. IC models include valid topological descriptions of heterogeneous objects and allow for the combination of cellular and functional representations of both the geometry of objects and their attributes. Here we provide a brief introduction to the IC framework; a more detailed and rigorous description can be found in (Kartasheva *et al.*, 2008).

Initially the concept of complexes was introduced in combinatorial topology (Alexandrov, 1998). Now complexes are actively used in computer graphics and computational geometry. Complexes provide a topologically correct description of various point set subdivisions and composite objects. In combinatorial topology, abstract and geometric complexes are distinguished. An abstract complex is a collection of abstract entities and relations between them. So “the abstract IC” is defined as a finite set of abstract elements called cells. A cardinal number called “dimensionality” is associated with each abstract cell. The entire set of the abstract ICs is subdivided into subsets of the cells of equal dimensionality. Then a finite set of binary relations on these subsets is defined. Note that the relations defined within the abstract IC do not have any geometric interpretation.

Thus, an abstract complex represents an abstract algebraic system. A geometric complex can be considered as an abstract complex associated with a collection of point sets in Euclidean space. In geometric complexes all the relations between the cells have geometric interpretations. An abstract complex provides a robust system for the description of relationships between subsets of composite objects. Using different associations of abstract complexes with geometric subsets one can construct topologically correct models of multi-component objects with different levels of detail of their relationships.

In (Kartasheva *et al.*, 2008) a detailed and rigorous definition of the IC

based model was developed for the representation of multi-component geometric objects with a full evaluation of mutual dispositions of its components. The IC concept extends the well known notation of cellular complexes. Compared with cellular complexes, ICs support additional relations between their cells - namely the containment relations. ICs are suitable for the representation of collections of overlapping cells.

The IC based model presented in (Kartasheva *et al.*, 2008) imposes very strong restrictions on the relations between IC cells. These restrictions are necessary for a full description of the mutual dispositions of all the point sets assigned to the cells. We will call such an IC model a cellular IC model. Cellular IC models are useful in various numerical applications (e.g. FEA, CAD and others). Note that in many applications of multi-component geometric objects, such as animation, visualisation, etc., not all the information regarding the mutual dispositions of the components is needed but some other dependencies of the components are required.

The geometry of an IC

First of all, let us present a definition of a geometric object represented in the IC representation. A geometric object G in Euclidean space contained in a hybrid model is the union of cells in this space:

$$G = \{g_1^{q_1}, \dots, g_n^{q_n}\} : g_i^{q_i} \subset E^3$$

where $g_i^{q_i}$ is a cell of dimensionality q_i . The point set enclosed in the cell $g_i^{q_i}$ is denoted as $|g_i^{q_i}|$. The following conditions must be satisfied for any object present in such model:

1. The boundary of each cell is the union of a finite number of cells of lower dimensions:

$$\partial g_i^{q_i} = \bigcup_{j=1}^N h_{ji} : h_{ji} = \begin{cases} g_j^{q_j}; q_j < q_i \\ \emptyset; q_j \geq q_i \end{cases}$$

2. Cells may overlap each other but the intersection of any two cells is

either the union of a finite number of cells or is empty:

$$g_i^{q_i} \cap g_j^{q_j} = \begin{cases} \bigcup_{k=1}^L g_k^{q_k} : k \in \{k_1, \dots, k_L\} \subset \{1..N\}; L \leq N \\ \emptyset \end{cases}$$

3. Each cell must be unambiguously defined by an existing geometric representation that allows us to perform a geometrically and topologically correct discretisation of the cell. It is important to note that only representations providing a way to convert into a mesh described by a polyhedral complex can be used. The ability to convert an IC into a mesh representation is usually required for the implementation of a range of traditional numerical methods and for topological analysis.

The topology of an IC

The topology of an IC model is described by a number of relations. The most important of these are:

1. The boundary relation, which is a relation between p-dimensional cells and s-dimensional cells of an IC ²:

$$Rb^{ps} = \bigcup_{i=1}^p \bigcup_{j=1}^s (g_i^p, g_j^s) : |g_j^s| \subset \partial |g_i^p| \wedge |g_j^s| \not\subset |g_i^p| \setminus \partial |g_i^p|; s < p$$

This type of relation allows us to track which lower-dimensional cells form the boundary of certain higher-dimensional cells.

2. The containment relation, which is a relation between p-dimensional cells and s-dimensional cells of an IC:

$$Rc^{ps} = \bigcup_{i=1}^p \bigcup_{j=1}^s (g_i^p, g_j^s) : |g_j^s| \subset |g_i^p| \wedge |g_j^s| \not\subset \partial |g_i^p|; s \leq p$$

The containment relation allows us to gain an understanding of the mutual dispositions of the cells enclosing other cells.

²The notation used here and further in the thesis is based on the notation used in (Kartasheva *et al.*, 2008)

Four additional relations based on the boundary and containment relations can also be introduced. These are the co-boundary, the “to be contained”, the incidence and the adjacency relations (Kartasheva *et al.*, 2008). A full description of a k -dimensional IC consists of a collection of cells and the corresponding relations between them:

$$K = \langle \bigcup_{i=1}^N g_i^{q_i}, \bigcup_{p=1}^k Rb^{p(p-1)}, \bigcup_{p=1}^k \bigcup_{s=1}^p Rc^{ps} \rangle = \langle G, Rb, Rc \rangle$$

The dimension of the IC is the maximal dimension of its cells.

As was mentioned earlier an IC can incorporate geometric models of different representations. Each IC cell can have a different geometric type. Five types of IC cells have been introduced thus far:

- The P-cell representing a polyhedron;
- The B-cell representing a k -dimensional manifold defined by its boundary (such as curve segments, surface patches and polygonal meshes);
- The F-cell representing an FRep model;
- The C-cell representing a composite cell which can aggregate cells of different types;
- The T-cell representing a constructive tree containing cells of various types mixed together using operations defined for IC cells.

Some of the provided definitions can be illustrated using a 2D example (fig. 12). This simple model consists of a rectangle represented by a B-cell and a disk represented by an F-cell. Apart from its geometry, the description of the model consists of a set of topological relations (fig. 13). From these relations we can see which of the 1D primitives create boundaries of 2D solid primitives. For instance, polyline **DEQS** is a boundary of a rectangle **DEQS**. These relations also reflect which cells are contained within other cells (which do not necessary need to be of a lower dimensionality).

The IC framework also defines set-theoretic and trimming operations on ICs. More details are provided in (Kartasheva *et al.*, 2008).

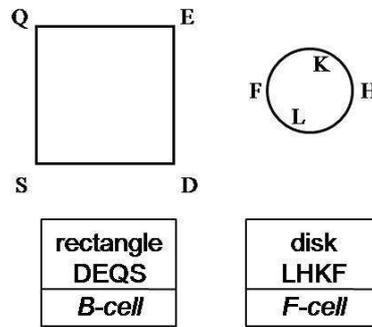


Figure 12: A set of cells present in the IC.

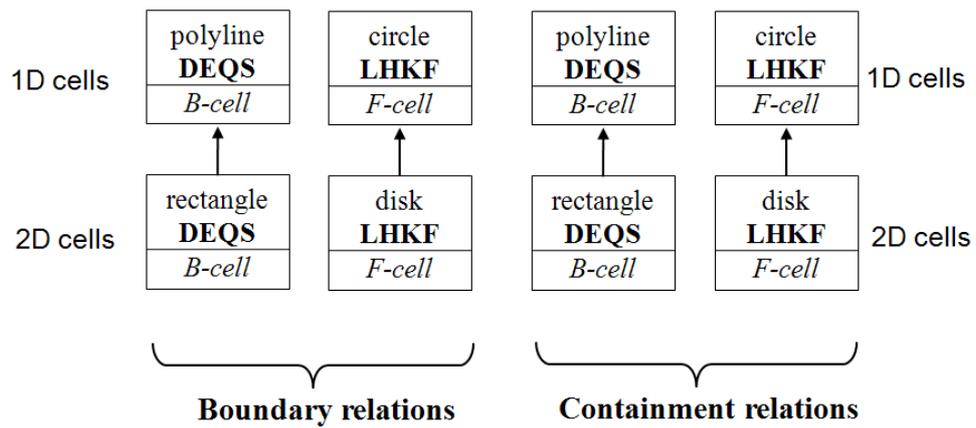


Figure 13: The topological relations of the IC.

From this description we can see that all the representations described earlier can be incorporated in an IC. This makes ICs suitable for a large set of modelling tasks and allows us to integrate various existing models as well as to define new complex hybrid models.

The attributes associated with an IC

The attributes representing different properties of a heterogeneous object are described independently of its geometry and topology. Each attribute Λ is described by a set of its values embedded into a multidimensional real number space \mathbf{R}^{m_Λ} , i.e. an attribute is a vector of dimensionality m_Λ that can be interpreted depending on the modelling context (e.g. colour, material, transparency, simulation parameters, etc). Any point within the modelling domain can be assigned a set of attributes using a collection of attribute functions:

$$S_\Lambda = \bigcup_{j=1}^J S_{\Lambda_j}; S_{\Lambda_j} : \Omega \rightarrow \mathbf{R}^{m_\Lambda}; \Omega \subset E^3$$

S_{Λ_j} is an attribute mapping, defined in a of section 2.2.3 ("Hypervolume extensions for FReps") which allows us to retrieve a scalar or a vector value of an attribute within the volume.

There are no specific constraints for attribute functions, which can be given in analytical form and can have discontinuities. It is important to note that the space-partitions associated with the attribute functions can in general differ from the geometric objects present in the IC. This means that another IC can be used for the description of the attributes of the model. Attribute relations are introduced in order to associate every cell of the IC with an attribute Λ :

$$Rs_\Lambda = \bigcup_{p=0}^3 Rs_\Lambda^p; Rs_\Lambda^p \subset G^p \times S_\Lambda$$

where G^p is a set of p -dimensional cells. Thus, if $(g_i^p, S_{\Lambda_j}) \in Rs_\Lambda^p$ the value of the attribute Λ at any point $X \in |g_i^p|$ is defined as $S_{\Lambda_j}(X)$.

The limitations of an IC framework

While the above IC framework goes a long way towards providing a unified way for model representation and manipulation, a number of unanswered

questions remain. One of these is the question of optimal conversion between different representations and the establishment of relations between these. In particular scenarios when precision is of high importance we might want to convert a BRep to an FRep (Fryazinov *et al.*, 2011). In other cases, it might be more appropriate to retrieve a polygonal representation of the FRep models. It is also important to introduce new techniques and additional improvements to make this model more flexible for the management of complex scenes and to provide opportunities for user manipulation. Additionally, dynamic hybrid modelling techniques need to be incorporated into ICs in order to make them suitable for the modelling of dynamic heterogeneous objects. These extensions to the framework will be described in detail in the next chapter.

Notwithstanding such questions, it is apparent that the IC framework is well suited for heterogeneous object modelling. Various representations supported by the IC framework make it possible to define multidimensional components of the objects using rich set of existing models. New models can be easily defined using a combination of the supported representations. The available set of topological relations allows us to provide additional important information describing the mutual dispositions of objects. The independent definition of an arbitrary set of attributes provides us with the flexibility of describing volumetric properties of multi-material objects. Altogether the IC framework provides us with a set of powerful tools for heterogeneous objects modelling. The main issue of concern with this framework is the absence of the user tools necessary for the modelling of dynamic objects. A set of methods has been developed over the years allowing us to define time-dependent models. These methods are suited for different types of models and problems being solved. In the following section we will outline the most common animation techniques which can be used for the definition of dynamic heterogeneous objects. These techniques need to be integrated into the IC framework.

2.4 A survey of computer animation techniques

In this section we outline a number of approaches commonly used for the definition of dynamic computer models. In a way similar to the discussion re-

garding the existing static model representations we analyse the strengths and weaknesses of the existing approaches for the definition of dynamic models. At the end of the chapter we discuss the necessity to extend the IC framework in order to use it for the definition of time-variant heterogeneous objects.

2.4.1 Keyframe-based animation

As was mentioned in the Introduction, we are interested in the modelling of natural heterogeneous objects. This means that static models alone, which we have been describing up to this point, are not adequate for the definition of time-varying objects. Even initially static heterogeneous objects could interact with other objects in a virtual scene. In this section we will examine existing techniques used for the definition of time-dependent models.

The transition from traditional animation to computer animation. Keyframing.

One of the most common and powerful ways of dynamic model definition is through the so-called “keyframing” and “inbetweening” technique. This technique is similar to the one used in traditional hand-drawn animation, where the production commonly starts with a set of storyboards. These storyboards reflect the key moments of the animation being produced (Thomas and Johnston, 1995). The next step is the preparation of a model sheet, which ensures the consistency of the animated entities present in the animation sequence. A senior animator then draws a necessary set of key drawings that are considered important for the animation (see fig. 14). After this step the inbetweeners add the intermediate frames “connecting” the key drawings. Most computer animation systems present the user with the same workflow, allowing the computer animator to define keyframes of the animation or more high-level key poses of the characters (see fig. 15). The inbetweening is done by the animation system. Such an approach provides the artist with a precise control over every aspect of the produced animation sequence.

In the early years of computer animation a lot of research was done in the area of 2D computer animation. It was important to automate the monotonous work produced by the inbetweeners. Back then, computers were only used for

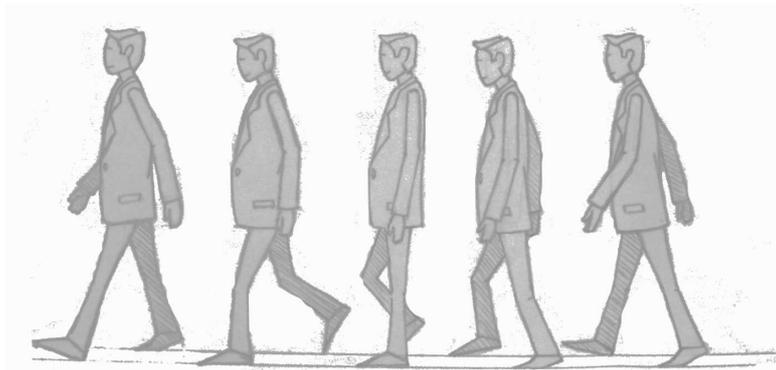


Figure 14: A set of keydrawings reflecting important stages of a walk-cycle (image courtesy of Jose Fonseca).

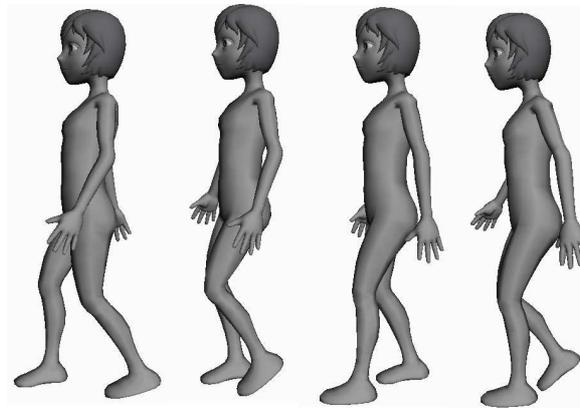


Figure 15: A set of key-poses defined by the computer animator (“Andy” model courtesy of John Doublestein).

some simple tasks in the production of 2D animations (for instance, scanning of hand drawings, colouring, background painting etc). Burtnyk and Wein (Burtnyk and Wein, 1976) proposed the use of time-dependent 2D skeletal deformations, allowing them to apply motion to static or dynamic 2D drawings, without the necessity to produce intermediate drawings³. Implementation of a reliable general inbetweening of 2D drawings appeared to be a very challenging task (Catmull, 1978). Edwin Catmull pointed out that 2D drawings were actually projections of 3D objects, which means that certain amount of information is lost at the projection stage. In order to restore the lost information additional information relating to the 3D appearance of the time-dependent object was required. One of the natural and most promising ways of providing this information was the definition of a time-dependent 3D

³The user still had to inbetween the skeleton into which the drawings were embedded.

model.

In a way similar to the CAD industry where line drawings were used for the visualisation of static objects (as described in section 2.1.1), early computer generated animations consisted of line drawings. As early as 1972 one of the first so called “half-tone” animations was produced (Catmull, 1972), in which the modelled object was represented by a set of shaded triangles. In this animation sequence the motion of a hand was defined using the hierarchical links between the various components of the hand. Time-dependent transformations could be controlled through a set of mathematical functions, producing certain motions over predefined intervals of time. The definition of a complex 3D sequence was still a rather laborious task. It was not until mid 80s when complex computer generated animation sequences could be produced by the artists rather than computer scientists. The application of fundamental principles used in traditional animation (Thomas and Johnston, 1995) to computer animation (Lasseter, 1987) was an important step in the history of computer graphics. Finally, artistic skills and knowledge applied to traditional animation could be transferred to and assisted by the computer. This allowed artists to have faster iteration times concentrating more on the creative side of the production, rather than on the unavoidable but necessary details of the animation. This change helped to improve the quality of the produced animations and to achieve effects that could not be produced in any other way.

The high level of control over dynamic entities and over various aspects of the sequence provided by the keyframe-based approach allows the artist to produce expressive and believable animation sequences. On the other hand, the definition of the keyframes and the set-up of the inbetweening parameters can still be a very arduous and time-consuming process. This process requires a lot of repetitive actions from the user over a number of iterations. Alternatively, major modifications of the sequence require numerous manual repetitive actions from the animator.

Although keyframing can be a very time consuming process, it is still one of the most commonly used animation techniques. For a large number of people it is easier to think of a dynamic model as of a set of static snapshots.

Each aspect of the snapshot can be adjusted over time, thus resulting in a time-dependent sequence with simplified control over the intermediate states of the model. Such an approach can be used to create fairly complex dynamic models with predefined behaviour. The majority of animators is used to this animation technique and find it very useful. If we wish to provide them with the ability to define time-dependent heterogeneous objects, keyframing needs to be supported by the new heterogeneous object modelling framework.

2.4.2 Procedural animation

In fact, historically a purely keyframed-based approach could not be easily used by the animators. Apart from the system described in (Burtnyk and Wein, 1976), there were no appropriate tools allowing the person without a specific knowledge of computer programming to define a dynamic model. On the other hand researchers producing computer animation sequences could provide a definition of a relatively simple process as a computer program. A number of high-level domain specific languages were developed (Hackathorn, 1977; Reynolds, 1982; Comninos and Webster, 1980; Fiume *et al.*, 1987). These languages provided special metaphors making it easier to direct the initially static entities in a relatively simple way. A high-level definition of the animation sequence can be seen as a form of screenplay; i.e. the creator does not need to define every single aspect of the process. Instead a high-level description is provided to the animation system. The animation system can then produce the actual frames of the animation sequence. In certain circumstances, complex evolution of a model could be more easily described through a scripting language incorporating high level terms used for animation production.

Let us refer to CGAL language (Comninos, 1986) illustrating the idea of a Domain Specific Language (DSL) oriented towards definition of an animated sequence. CGAL is a Pascal-like language (Wirth, 1971). It provides such abstractions as affine transformations of the objects, camera, different types of light sources, shapes and deformations, keyframes, events, procedural geometric modelling etc. One of the important concepts of CGAL is the concept of events, which allow us to perform certain actions only within specified time

intervals, i.e. time-dependency coupled to imperative statements is inherent to the language. This is similar to a detailed film script, where every action happens at certain time. This allows us to perform tight synchronisation of various events occurring in the model and to precisely plan the timing of all the actions.

Here is a simple example demonstrating definition of a basic animation:

```
(* A typeless definition of an object, duration of a sequence *)
(* and a scale event *)
var object, duration, scaleDuration;
begin
  object := inof 'mesh.obj'; (* load external mesh *)
  duration := 60; (* the sequence will take 60 frames *)
  scaleDuration := 40; (* scaling will take 40 frames *)
  ...(* set-up a model *)
  script 1 to duration do
  begin
    (* actions defined inside this block will only take *)
    (* place from 1st to 60th frame*)
    (* event A: shift the object along x between frames 1 and 20*)
    from 1 to 20 do tx {object} 1;
    (* event B: rotate the object around z between frames *)
    (* 10 and 30*)
    from 10 to 30 do rz {object} 15/20;
    (* event scale: scale the object for scaleDuration frames*)
    from 20 to 20+scaleDuration do sc {object} 1.1 1.1 1.1;
  end;
end.
```

The description of a model does not require any low-level definitions and initialisation code, which would be mandatory in a general purpose programming language. The user works with high-level 3D modelling and animation metaphors, concentrating on essential aspects of the model that can easily be modified. From this example we can see that a user can define a set of frames he wants to work with. Within these frames he or she can describe events occurring simultaneously or sequentially. The starting and finishing times and the duration of events can be defined in a parametric manner, which allows the user to easily adjust the animation sequence. We should take advantage of this flexibility of model definition in the system for heterogeneous objects modelling. A high-level DSL incorporating concepts specific to time-dependent hybrid models could simplify the process of model definition and make the modelling itself more accessible and efficient.

May introduced the notion of Encapsulated models in (May, 1998). These

models contain a set of attributes including but not limited to shape, motion surface properties, user interfaces (UI) for controlling the model and sounds. All these attributes can be described using a procedural definition. A special DSL called “AL” was developed in order to provide these high-level procedural definitions of static and dynamic models. The main focus of the approach proposed by May was the creation of interactive tools available to the end user. These visual tools could be defined or parameterised by a more experienced user through the “articulation functions” implemented in “AL”. This method provided a multi-level tool for model definition, which allowed the user to manipulate complex models in real-time.

Another interesting approach to modelling the natural world is so called “Empirical Modelling” (Beynon, 1987). Empirical models do not require us to define a precise physical model of the process we want to simulate. We can define a simplified model of the phenomenon with a number of available parameters. In the process of interaction with the model through the provided parameters, the model can be further refined or adjusted. This allows us to reproduce in our own model the behaviour of the system that is being modelled. Empirical modelling has been used for the definition of geometric models (Beynon and Cartwright, 1989), where the constructive history of the modelled object could be defined using a symbolic description incorporating different geometric representations. In theory this allows the user to define complex geometric objects of different representations and relations between them within a single framework.

Generally a procedural approach is more flexible and extendable as it allows us to incorporate a large set of problems that can be solved using physically inspired models of the real world. Certain types of animation involving simulation of natural phenomena are next to impossible to produce using keyframed-based approaches. For instance, rigid-body simulation (Witkin and Baraff, 1997), particle-based systems (Reeves, 1983) for fluid simulations (Monaghan, 1988), dynamic simulation of natural plants (Prusinkiewicz, 1986), cloth simulation (Terzopoulos *et al.*, 1987) and many more. Overall this approach becomes indispensable when the animator is interested in the verisimilitude of the modelled event or phenomenon. Witkin and Kass (Witkin and Kass, 1988) proposed a hybrid approach which allows

the animator to define certain constraints for the animated model similar to rough keyframes. Their system then finds a solution satisfying the specified constraints and produces an animation sequence that looks physically correct.

A procedural definition assumes the existence and development of a conceptual model of the event or phenomenon. This implies that the modeller makes an effort to understand the underlying process and adequately describe it, rather than to simply reproduce it. This is a very important aspect of the heterogeneous objects modelling system. As our main aim is not to solely reproduce the observed phenomenon (which can be a valid modelling task too), but a desire to provide a modeller with a platform which can be used for the definition of an accurate physical model of time-dependent heterogeneous objects. The behaviour of this model can then be simulated according to specific requirements. This would allow us to produce a new system for the dynamic modelling of heterogeneous objects of varying complexity and will allow the user to interact with such objects. This is different compared to mimicking and predefining the dynamic characteristics of heterogeneous objects. Unlike in keyframed-based animation, the behaviour of the defined model may not be known in advance. An iterative definition of standalone components of the model allows the user to combine different objects and states of the model together, resulting in new behaviours. Kalra and Barr proposed a similar concept allowing them to introduce these features to the modelling system (Kalra and Barr, 1992). In this type of models it is often natural to operate in terms of such metaphors as events. Events occur in the model over time and result in transitions between the states of the model in an order that is not known in advance (i.e. event-driven animation). Another powerful feature of this approach is the possible presence of dependency relations between the entities of the model. This allows the user to define a set of independent entities and their behaviour in a modular fashion and to combine these, thus establishing dependencies between their properties. For instance, the LSD-engine (Adzhiev and Beynon, 1999) allows the user to experiment with a multicomponent interactive model and to refine it on the fly in a stepwise manner.

Events provide semantic information about the model. A sequence of events can be used to determine the current state of the model and to help

us gain a better understanding of the intermediate phases that the model went through. Initiating reactions and behaviours depending on the occurrence of events is a natural way of thinking for modellers, as it is similar to their inherent thinking process. This makes modelling more accessible and intuitive. Events reflect certain critical points in the simulation process, which provides the user with meaningful information about the model over time. Besides, it helps to approach modelling in a systematic manner, subdividing possible states and composing them together in an unlimited number of ways.

Although procedural model definition is a very powerful way of dynamic model definition it has certain limitations. First of all, even an iterative procedural definition of a model can be a very tedious and error-prone method. There is no formal way to define a model in a certain way. A valid and adequate model definition may require a high-level of expertise from the user. The procedural definition of a model requires an experienced modeller with a solid technical background. Secondly, some types of animation require strong artistic control and cannot be appropriately defined in a procedural manner. For instance, to date a complex walk cycle animation of a biped character or a believable facial animation cannot be produced without the assistance of an experienced artist. It is quite common to incorporate certain low-level animation sequences into the procedural definition of a model. This can only be done through the combined efforts of both an artistic and a technical specialist.

Overall, we can see that the procedural approach to dynamic objects modelling is a rather powerful and versatile technique. It is very important to support this method of model definition in the dynamic heterogeneous objects modelling framework, as we are interested in providing the means of defining non-deterministic systems with complex behaviours. This can be an Empirical model of a time-dependent heterogeneous object, refined and interacted with on the fly, or by the physical simulation of a natural process that allows us to describe the behaviour of some complex phenomenon. The procedural approach offers us the flexibility to define dynamic models in a number of different ways, without significant constraints on the type of model being described. Unlike the keyframed time-dependent model, a procedural definition allows us to retrieve new results and behaviours based on the initial

description of the model without the necessity to redefine the key states of the time-dependent model.

2.4.3 Conclusions

From the above discussion it should be apparent that, similar to static object modelling, time-dependent modelling does not have an ultimate technique that can be used for the definition of different models. Both approaches are suited to the solution of dissimilar problems and need to be intermixed. The combination of methods described in this section allows us to get the best of both methods, having precise artistic control over some parts of the model, while providing a high-level definition of other parts of the model that can be evaluated on the fly.

Earlier in this chapter we described the existing approaches to heterogeneous objects modelling. Current research is mostly centred around static models. Dynamic hybrid models are still an on-going research topic (Xi-aoping and Dutta, 2003; Adzhiev *et al.*, 2002) with a large set of unanswered questions. As Kou and Tan (Kou and Tan, 2007) note, dynamic heterogeneity modelling can have a wide range of applications in such areas as biomedicine, surgery simulation, mechanical engineering and others.

In this research project we wish to combine the currently available techniques for static heterogeneous objects modelling, incorporating a number of the existing representations, with a set of methods used for the definition of time-variant models. In order to achieve this, the currently available IC framework needs to be extended. The proposed extensions to the framework are described in the next chapter.

3 Theoretical framework for Dynamic Implicit Complexes

In this chapter we introduce a new dynamic IC framework allowing us to deal with mixed-dimensional hybrid models whose structure and properties change over time.

3.1 Motivation

In previous chapters we have outlined a number of the existing representations and methods used for the definition of time-dependent models. Each representation and dynamic model definition has a number of strengths and weaknesses. There have often been good reasons for some of these methods to be used in various applications over the years. There is no compelling reason to abandon previous approaches in favour of a new one. Preferably we should be able to accommodate the entire set of existing representations. We need to introduce a new framework allowing us to incorporate existing representations while overcoming their existing limitations. The IC framework described in the previous chapter allows us to combine models of a diverse nature within one hybrid model. Unfortunately, to date the IC framework was only suited for the definition of static hybrid models. Thus, it could only be used for a limited set of applications. Here we introduce a new dynamic IC framework allowing us to deal with time-dependent hybrid models along with the production of their corresponding animations. This new framework is partly based on the static IC framework. Our new dynamic framework provides a means to describe complex behaviours of a model in a relatively simple way. This is achieved through the combination of procedural time-dependent model definitions, based on event-driven dynamics, with widely-used traditional keyframe-based approaches.

Next we provide an extensive description of the components of the framework together with some simple examples.

3.2 Dynamic IC cells

First, we need to introduce the concept of time within the IC framework in order to allow us to define an appropriate behaviour for the entities present in the model and to describe both their structural and parametric changes over time. A basic set of the dynamic IC definitions is as follows:

- One of the basic concepts required for the definition of a dynamic model is that of a time span:

$$T_i = \bigcup_j^{N_i} (t_{a_j}^i, t_{d_j}^i) \Big|_{St_i}; t_{a_j}^i, t_{d_j}^i \in E$$

where $t_{a_j}^i$ and $t_{d_j}^i$ define the start and end times of the span accordingly and St_i is the scale factor of this time span, i.e. it defines the rate at which time changes within the span. Unique values of St_i allow us to define local time spaces and to measure arbitrary time intervals.

- Each cell present in the model has a life span associated with it. The life span of a cell is defined in a way similar to the time span:

$$L_i = T_i = \bigcup_j^{N_i} (t_{a_j}^i, t_{d_j}^i) \Big|_{St_i}; t_{a_j}^i, t_{d_j}^i \in E$$

where $t_{a_j}^i$ and $t_{d_j}^i$ define the activation and deactivation times of an entity. Within the life-span, every active entity has access to both its local time and the global time associated with the entire IC. In certain cases the life-span of entities can be evaluated on the fly during the process of the model evaluation, thus providing a way for dynamic model modification.

- To take into account the time dependency of the cells we consider the point set of each cell as a function of time $g_i^{q_i} = g_i^{q_i}(t)$. We define the initial point set $g_i^{q_i}(0)$ of a cell and its bounding domain $D_i^{q_i}(t) \subset E^n$, such that $g_i^{q_i}(t) \subset D_i^{q_i}(t)$ for any given time t . To describe a dynamic point set $g_i^{q_i}(t)$ we introduce the following functions:

- The shape function $H(t) : E^n \times T \rightarrow E^n$, which defines a point set. For each time moment the shape function of a cell gives a representation of the cell shape in a form corresponding to the cell type;
- The deformation function $W(t) : E^n \times T \rightarrow E^n$, which modifies a point set. This function provides descriptions of various deformations of a geometric object (for instance, this can be a bend operation, a taper operation or a more complex non-linear space mapping operation). Deformations are applied to the geometry of the cell in its local space and perform an arbitrary space mapping of the initial point set;
- The motion function $M(t) : E^n \times T \rightarrow E^n$, which represents an affine transformation of a point set. A time-dependent affine transformation $M(t)$ allows us to define the mapping of the initial point set of a cell from its local coordinate system to the global modelling space.

Thus, a dynamic point set $g_i^{q_i}(t)$ is defined as follows:

$$g_i^{q_i}(t) := \{g_i^{q_i}(0), H_i^{q_i}(t), M_i^{q_i}(t), W_i^r(t), D_i^r(t), L_i\}$$

where L_i is the life-span of the cell.

In the definition of a particular geometric cell some of its components may be omitted. For example, we could describe a dynamic set using a shape function alone or using an initial set and a motion applied to it, and so on.

We also introduce global parameters that are defined in the frame of the object and in the global life-span of the object. Global parameters and global time are used for event-driven control of the cells and their synchronisation in the frame of a multi-component object.

- Apart from the set of predefined properties mentioned above, each cell can be assigned a set of arbitrary parameters meaningful within the context of the cell (e.g., radius or density). The “local” parameters are

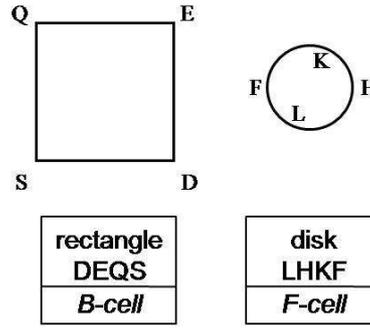


Figure 16: A set of cells present in the IC.

only available within the scope of the cell while the “global” parameters can be exposed and can be made available outside the cell. These parameters are defined as expressions involving other parameters, thus:

$$P_c^i(t) = \left\{ \bigcup_j^{N_{PG}^i} p_{G_j}^i(t), \bigcup_j^{N_{PL}^i} p_{L_j}^i(t) \right\};$$

where $p_{G_j}^i$ is the j -th global parameter and $p_{L_j}^i$ is the j -th local parameter of the i -th cell. These are the predefined types of parameters (scalar, vector, set of polyhedrons, etc). The values of the parameters over time are either defined by a set of entity reactions or through the assignment of a predefined animation curve. This allows us to define complex dynamic dependencies between parameters associated with different entities (see section 3.5).

To make our description of the dynamic point sets more flexible, we introduce a parameterisation of all the components. We define a parameterised time-dependent point set in the following way:

$$g_i^{q_i}(t, P_c^i(t)) = \{g_i^{q_i}(0, P_c^i(t)), H_i^{q_i}(t, P_c^i(t)), M_i^{q_i}(t, P_c^i(t)), W_i^r(t, P_c^i(t)), D_i^{q_i}(t, P_c^i(t)), L_i(P_c^i(t))\}$$

For every time moment the proposed dynamic model of a geometric cell provides the corresponding values of the parameters and the resulting point set associated with this cell.

We will now refer to a simple example shown in figure 16. We have

used this example earlier in the text to demonstrate a simple IC mode. Now we can define the motion of the disk $LHKF$ over time (see fig. 17).

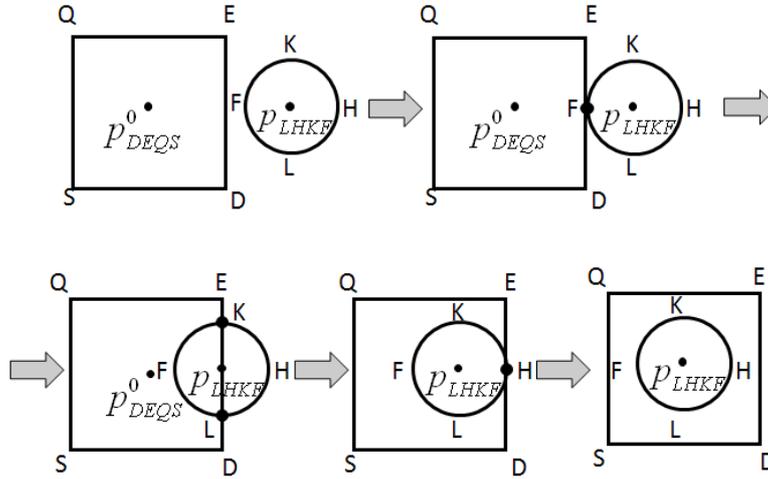


Figure 17: The motion of the disk defined over time.

The translation transformation of the disk $LHKF$ is defined by a set of parameters:

$$p_{LHKF}(t) = p^0_{LHKF} + v_{LHKF} \cdot t$$

where p^0_{LHKF} is the initial position of the centre of the disk (this initial position can be defined using $M_i(t)$) and v_{LHKF} is the velocity of the disk, which is a custom parameter added by the user to the set $P^i_c(t)$. In this example we assume that the rectangle $DEQS$ is a static cell with its centre at the point p^0_{DEQS} (i.e. its $M_i(t)$ is constant over time).

3.3 Dynamic IC attributes

Attributes play an important part in the definition of a heterogeneous object.

As in the static case, attributes representing the properties of heterogeneous objects are described independently of its geometry and its topology. Attribute vectors are now defined using time dependent mappings:

$$S_{\Lambda}(X, t) = \bigcup_{j=1}^J S_{\Lambda_j}(\tilde{X}_{\Lambda_j}(X, t), t); X \in \Omega$$

$$S_{\Lambda_j}(\tilde{X}_{\Lambda_j}(X, t), t) : \Omega \times T_{\Lambda_j} \rightarrow \mathbb{R}^{m_{\Lambda}}; \Omega \subset E^N;$$

$$\tilde{X}_{\Lambda_j}(X, t) : \Omega \times T_{\Lambda_j} \rightarrow \Omega$$

where the attribute Λ_j is a vector of dimensionality m_{Λ} , S_{Λ_j} is a time-dependent attribute mapping, T_{Λ_j} is the life span of the attribute Λ_j and $\tilde{X}_{\Lambda_j}(X, t)$ is a time-dependent space mapping.

As in the case of cells, a point in space is first mapped to the global space using a deformation and an affine transformation:

$$\tilde{X}_{\Lambda_j}(X, t) = \mathbf{M}_{\Lambda_j}(t) \cdot W_{\Lambda_j}(X, t) : E^N \times T_{\Lambda_j} \rightarrow E^N$$

The mapping of the input point-set is required in order to provide a way of linking the time-dependent properties of the cells with its attributes. For instance, the attributes can be defined in the local space of a dynamic cell (with its geometry acting as a dynamic space partition) that virtually follows its motion. Figure 18 illustrates this idea by using a simple taper deformation applied to a static geometry and its attributes.

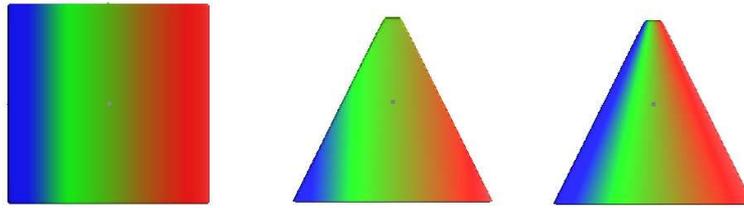


Figure 18: A space mapping used for attributes: (left) initial geometry and attributes; (centre) deformed geometry and initial attributes (right) the same deformation applied to geometry and attributes simultaneously.

A dynamic attribute mapping associated with a space mapping is defined by the dimensionality of the attribute, its attribute mapping, its time-dependent transformation and deformation, a set of attributes and the lifetime of the at-

tribute

$$A_{\Lambda}(X, t) = \{m_{\Lambda}, S_{\Lambda}(X, t), W_{\Lambda}(X, t), \mathbf{M}_{\Lambda}(t), L_{\Lambda}\}$$

In a way similar to cells, attributes can have a set of custom parameters allowing us to perform their parameterisation:

$$A_{\Lambda}(X, t, P_c(t)) = \{m_{\Lambda}, S_{\Lambda}(X, t, P_c(t)), W_{\Lambda}(X, t, P_c(t)), \mathbf{M}_{\Lambda}(t, P_c(t)), L_{\Lambda}(t, P_c(t))\}$$

As was shown in fig. 18, this mapping is useful for the association of a dynamic geometry with its attributes.

3.4 Extensions to FRep within the dynamic IC framework

We have already outlined the basic concepts needed for the definition of a dynamic cell within the IC framework. We have also mentioned that the Constructive Hypervolume Framework (CHF) provides a powerful way for heterogeneous object modelling. In order to accommodate Hypervolume objects in IC cells, we need to introduce certain extensions to the CHF. These extensions will allow us to simplify the integration of heterogeneous models, defined using FReps, into our hybrid mixed-dimensional modelling framework. We will describe the extensions introduced in this thesis in the following sections.

3.4.1 Modelling domains

Currently the conceptual model of FReps is very broad. It can support multi-dimensional modelling functions, thus allowing us to create models of various dimensionalities according to the definitions in section 2.2.3:

$$f(P) : E^n \rightarrow E$$

For instance, FReps allow us to define a 2D object on a plane or 4D models in a space-time domain (Fausett *et al.*, 2000; Pasko *et al.*, 2004a). However at the moment the FRep library is mostly oriented towards the creation of

3D models. Most of the available primitives and operations are defined in 3D Euclidean space. The creation of models of mixed dimensionalities can be a complex and laborious task. Additionally, FRep objects present in this model can not be classified according to their modelling space and dimensionality, which may lead to cases of superposition of incompatible entities⁴. It is highly desirable to explicitly introduce the concept of permitted modelling spaces. This is an important requirement for the correct definition of mixed-dimensional models. Additionally a set of primitives and operations available for the creation of complex multidimensional models would greatly enhance the modelling capabilities of FReps and increase the number of areas where FReps could be applied.

In order to correctly accommodate FRep defining functions within IC cells of different dimensionalities we need to explicitly define the domain of the defining function: $f_n(P) : E^n \rightarrow E$. When using FRep objects inside F-cells within the dynamic IC framework we can use this domain information to match FRep entities to IC cells:

$$g_i^{q_i}(t) : H_i^{q_i}(t) = f_n(P) : E^n \rightarrow E; q_i = n$$

A modelling domain is a specific set over which an FRep model is defined. Each modelling domain has a specific dimensionality associated with it. Domains available in an FRep model are listed in table 1.

Domain	Dimensionality	Available coordinates
X	1D	(x)
T	1D	(t)
XY	2D	(x, y)
XT	2D	(x, t)
XYZ	3D	(x, y, z)
XYT	3D	(x, y, t)
XYZT	4D	(x, y, z, t)

Table 1: *The list of available FRep modelling domains.*

The domain of an IC cell is then a subset of the modelling domain of an

⁴For instance, this may be a CSG operation between a 2D circle and a 3D block object.

FRep defining function:

$$D_i^{q_i}(t) \subset E^n; q_i = n$$

As can be seen from table 1, a predefined set of available domains provides the option to create models in seven domains from 1D to 4D. “Multimedia domains” of higher dimensionalities (Fausett *et al.*, 2000) can be introduced via arbitrary additional parameters existing within the model. Each modelling domain provides a specific number of variables that are available for the definition of the shape or attributes of a model. Time t is present in the model explicitly (see section 3.4.2). Particular types of primitives and operations can access and modify t , while static geometric entities can only get access to a number of geometric coordinates according to their dimensionality.

In the general case, an FRep model can be constructed from objects of different dimensionalities and we need to correctly establish relationships between these. Hence, each FRep entity has a specific input and output domain. Some entities of higher dimensionality can use entities of lower dimensionality for their definition and vice versa. Specific operations designed to change the dimensionality of an object should be available as well. For instance, we may need to project an object to a lower-dimensional space or to construct an object of a higher dimensionality.

In some situations, a lower dimensional primitive may be expected to be defined in one of the domains which are not available (e.g. (y) , (y,z) or (x,z)). This may be the case for particular FRep entities constructed from lower dimensional primitives (such as the “Cartesian product” used as an example later in this section). In this case, a “reduce dimensionality” operation can be used. This operation performs a mapping of a specified set of coordinates to another set of coordinates:

$$g_n(P) : E^n \rightarrow E^m; m < n$$

This operation can be interpreted as a “re-projection” of a higher dimensional coordinate set to one of the existing lower dimensional domains. Such an operation allows the user to define all the lower dimensional FRep entities in

one of the predefined domains and to transfer the resulting object to another domain required by the higher dimensional primitive (fig. 19).

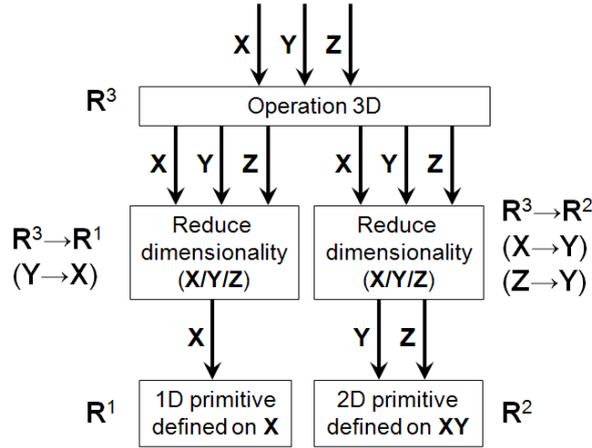


Figure 19: The “reduce dimensionality” operation.

This operation can be illustrated by the example of a 3D “Cartesian product” operation. Such an operation may require a 2D primitive to be defined on the **XZ**-plane and a 1D primitive on the **Y**-axis (fig. 20). The initial 2D shape is defined on the **XY**-plane and then transferred to the **XZ**-plane. A line segment is defined on the **X**-axis and transferred to the **Y**-axis.

At the same time a Cartesian product can be considered as an operation increasing the dimensionality of the entity ($g_n(P) : E^n \rightarrow E^m; m > n$). This operation produces a new object with a dimensionality higher than the dimensionalities of both entities used as input to this operation. A new set of built-in primitives and operations for all of the domains will be introduced in the next chapter.

F-cells can be used for the definition of heterogeneous objects of different dimensionalities. We can see that the explicit introduction of modelling domains allows us to build complex multi-dimensional objects used for the definition of IC cells and attributes.

3.4.2 Space-time

In simple models affine transformations or global deformations changing over time are applied to static geometric objects. Such an approach can sometimes

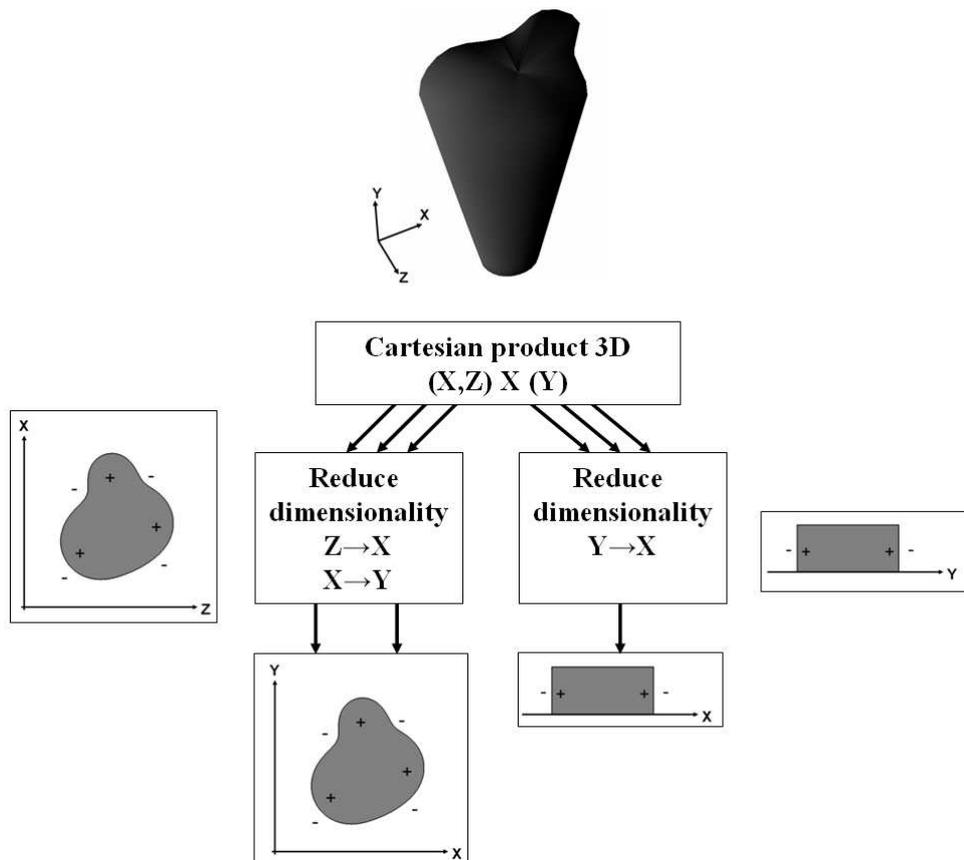


Figure 20: Meeting the requirements of a 3D Cartesian product.

be limiting, especially when dramatic changes to the model occur over time. Some of the existing time-dependent FRep models were animated with the introduction of general “multimedia” coordinates (Fausett *et al.*, 2000; Pasko *et al.*, 2004b), which were changed independently of the model. The explicit introduction of time to the model (see previous subsection) introduces new concepts and FRep objects. One of our main aims is to simplify the process of dynamic model definition and to provide new opportunities for modelling in the space-time domain.

If time is present in an FRep model, it can be manipulated in the same way as any other geometric coordinate (see section 3.4.1). The explicit presence of time in the model allows the user to define a geometric model and to establish complex dependencies between objects over time. An object defined in the space-time domain can be interpreted as the union of lower dimensional geometric objects defined on a set of space-time hyper-planes. Thus,

a custom projection of a time-dependent object onto a lower dimension (i.e. onto a geometric space) can be used to specify a geometric object and vice versa. Thus, one can imagine a geometric FRep object defined on a plane and a “profiling” object defined on a space-time plane (see lower portion of fig. 21a). The intersection of extrusions (or sweeps along an infinite line segment parallel to one of the axes) performed in orthogonal hyper-planes generates a 3D space-time object (see fig. 21a). This object can be thought of as a time-dependent 2D object, whose deformation is specified by a “profiling” projection defined in the space-time hyper-plane (see fig. 21b)

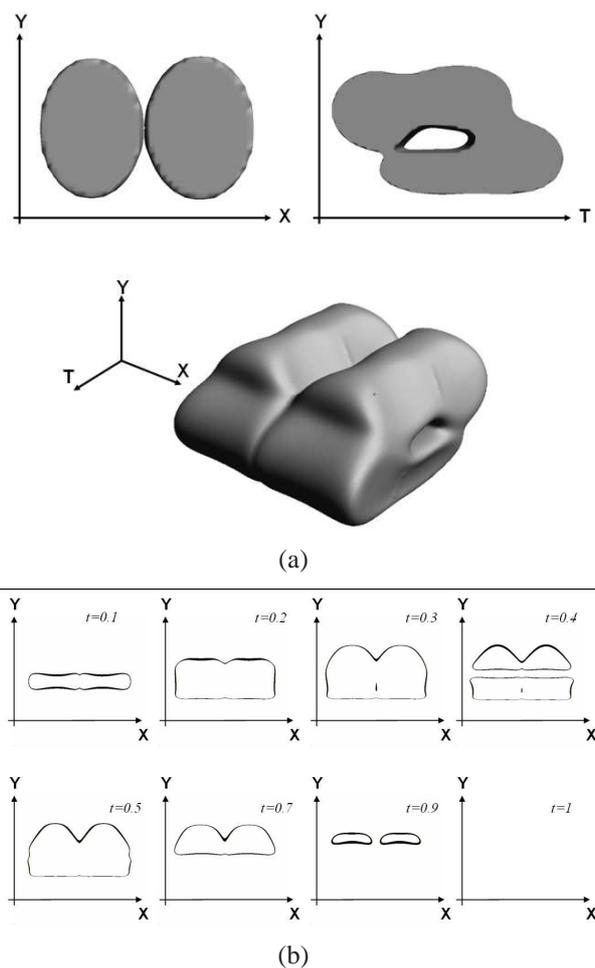


Figure 21: Space-time object: (a) Two defining 2D projections (b) A set of 2D geometric slices of the resulting space-time object.

Another example would be that of space-time blending (Pasko *et al.*, 2004a, 2010). This type of operation allows us to perform advanced blending operations in higher-dimensional space. Blended objects are defined in a purely

geometric domain (see fig. 22), while their higher dimensional “prototypes” are blended in a space-time domain of a higher dimensionality. The resulting higher-dimensional object is shown in fig. 23. Slices of this space-time object put together can be used to achieve smooth transition between complex 2D or 3D objects. We provide more information on this operation and introduce a number of improvements to this operation later on in this thesis.

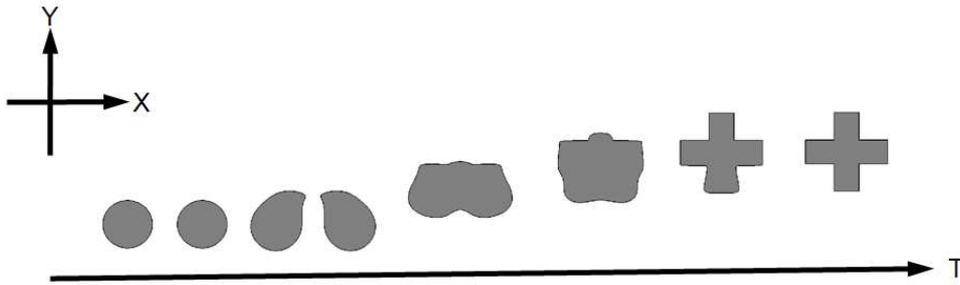


Figure 22: A set of 2D cross-sections of the objects metamorphosed over time.

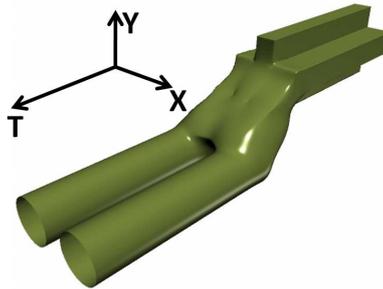


Figure 23: A 3D objects defined in the space-time domain.

This operation is similar to the domain switching operations defined in previous sections. This operation allows us to easily switch between purely geometric domains and space-time domains. The explicit presence of time allows us to define the time-dependent components of dynamic heterogeneous objects through F-cells. The availability of time in the IC framework, introduced to the F-cells, allows us to keep all the entities of our hybrid model in sync. These extensions are crucial for the correct integration of FReps into the dynamic IC framework and for the definition of valid hybrid models.

3.5 Dynamic IC relations

From the extensions presented in the previous section we can conclude that all the relations defined in an IC become dynamic as well. This means that relations may be established and removed over the course of the modelling process. Depending on the mutual locations of the cells and the modifications to their geometry any topological relation of the IC can become invalid. Thus, it should be possible to provide a description of the topological relations of the model within different time intervals. As was mentioned earlier in the text (see section 3.5), the explicit enumeration of the topological relations or their automatic establishment can be omitted, if they are not considered to be crucial to the definition of the model.

The static IC framework did not provide an explicit dependency relation between the cells or attributes present in the model. The simplest form of a dependency relation is that of a relation between a “master entity” and a “dependent entity”. Any modification of the state of the master entity affects the state of the dependent entity⁵. Here we introduce additional dependency relations to the framework:

1. The establishment of the dependency relations between the appropriate parameters of the cells/attributes leads to the implicitly defined dependency relations between the cells/attributes. More formally, two IC cells/attributes belong to a dependency relation, if any parameter of the cell/attribute C_j is defined using any parameter or set of parameters of the cell/attribute C_i :

$$Rd_p = \bigcup_{i \in I_m^{i,p}} \bigcup_{j \in I_d^{i,p}} (C_i, C_j) |_{p_{ij}^{Rdp}} \quad (3)$$

where p_{ij}^{Rdp} is the priority of the dependency relation between the cells/attributes i and j , $I_m^{i,p}$ is the set of indices of the master cells/attributes and $I_d^{i,p}$ is the set of indices of the cells/attributes dependent on the cell/attribute C_i . Priorities provide an additional tool that can be used for the resolution of certain model evaluation ambiguities. The values

⁵A detailed description of the states is provided in section 3.7.

of the priorities associated with dependencies can be employed by the user in order to provide information about the desired evaluation order of the entities present in the model. Overall, parametric dependencies provide a powerful way for model parameterisation.

Parametric dependencies can be used to define both the geometry and attributes of an IC entity. But we need to introduce dependencies between the geometry and the attributes in order to provide a flexible way of defining more complex relations between the IC entities.

2. Another dependency relation that is established in the framework is the dependency between the geometry of the cells:

$$Rd_g = \bigcup_{i \in I_m^{i,g}} \bigcup_{j \in I_d^{i,g}} (C_i, C_j) |_{p_{ij}^{Rd_g}}$$

where $p_{ij}^{Rd_g}$ is the priority of the dependency relation between the cells i and j , $I_m^{i,g}$ is the set of indices of the master cells and $I_d^{i,g}$ is the set of indices of the cells dependent on the cell C_i . In this case, the state of a dependent cell can be modified using the geometry of the master cell. This type of dependencies is important for geometry manipulation and for the definition of complex deformations within the IC framework.

3. We also introduce the dependency relation between attributes, in order to be able to define complex dependencies between dynamic attributes:

$$Rd_a = \bigcup_{i \in I_m^{i,a}} \bigcup_{j \in I_d^{i,a}} (H_{\Lambda i}, H_{\Lambda j}) |_{p_{ij}^{Rd_a}}$$

where $p_{ij}^{Rd_a}$ is the priority of the dependency relation between the attribute entities $H_{\Lambda i}$ and $H_{\Lambda j}$, $I_m^{i,a}$ is the set of indices of the master attributes and $I_d^{i,a}$ is the set of indices of the attributes dependent on the attribute entity $H_{\Lambda i}$. This type of dependency can be used for the definition of compound attributes composed of simpler attribute definitions.

4. Next, we introduce the relation establishing dependencies between the

geometry and the attributes:

$$Rd_{ga} = \bigcup_{i \in I_m^{i,g}} \bigcup_{j \in I_d^{i,a}} (C_i, H_{\Lambda_j})|_{p_{ij}^{Rd_{ga}}}$$

where $p_{ij}^{Rd_{ga}}$ is the priority of the dependency relation between the geometry of cell C_i and attribute H_{Λ_j} , $I_m^{i,g}$ is the set of indices of the master cells and $I_d^{i,a}$ is the set of indices of the attributes dependent on the cell C_i . This type of dependency can be used for the definition of attributes based on geometry. For instance, this could be the definition of a complex space partition defining the values of the attribute in the volume.

5. Finally, we introduce the relation establishing dependencies between the attributes and the geometry:

$$Rd_{ag} = \bigcup_{i \in I_m^{i,a}} \bigcup_{j \in I_d^{i,g}} (H_{\Lambda_i}, C_j)|_{p_{ij}^{Rd_{ag}}}$$

where $p_{ij}^{Rd_{ag}}$ is the priority of the dependency relation between the attribute H_{Λ_i} and the geometry of the cell C_j , $I_m^{i,a}$ is the set of indices of the master attributes and $I_d^{i,g}$ is the set of indices of the geometric cells dependent on the cell attribute H_{Λ_i} . This type of dependency can be used for the definition of geometry based on attributes. For instance, this dependency can be used for the description of material-aware deformations (Popa *et al.*, 2006), where the influence of a deformation is evaluated using the attributes associated with the shape of the object.

The generic set of dependency relations is defined as a superposition of all the aforementioned relations:

$$Rd = Rd_p \cup Rd_g \cup Rd_a \cup Rd_{ga} \cup Rd_{ag}$$

Any of the master entities can, in its turn, depend on a set of other entities, i.e. $i \in I_m^*; \exists j : i \in I_d^{j,*}; i \neq j$ is valid. This means that we can combine dependency relations and build a complex dependency graph. This allows the user to describe sophisticated models using relatively simple building blocks

and to provide the description of connections between them. Additionally, this allows us to localise the behaviour of the cell within its reaction function (see section 3.7). We can then dynamically modify the dependencies of the cell without actually modifying the reactions of the cell (i.e. only the dependency need be changed, not the reaction procedure of the cell). This can also simplify iterative work on the model, when the user “clones” pre-configured cells and only modifies the context they reside in via the introduction of new dependency relations⁶. Thus each property is treated as a custom “interface” of the cell to the “outer world”. The establishment of dependencies between the properties of the cells is similar to the connection of the appropriate interfaces.

We also provide the user with a predefined dependency relation called the hierarchical dependency. This relation reflects the dependency between the transformations of the cells, automatically performing evaluations of the child transformation based on the up-to-date transformation of the parent. This is a very common and important relation in computer animation, greatly simplifying the definition of a large set of natural models. Dependency relations can also be used for the definition of complex deformations. In this case the master cell defines the initial geometry, while a dependent cell applies a deformation to this geometry. The resulting deformed shape becomes available through the geometry of the dependent cell.

Let us now consider the specifics of the geometric interpretation of the relations within the dynamic IC framework. The boundary relation relates cells of different dimensionalities. If the pair of abstract cells (g_i^p, g_j^r) belongs to the boundary relation, this means that the point set $|g_j^r|$, corresponding to the abstract cell g_j^r , belongs to the boundary of $|g_i^p|$, which itself corresponds to the abstract cell g_i^p . Note that the inverse condition is not required for the framework to be consistent. In other words, for some point sets there are no established pairs in the form of the abstract boundary relation despite the fact that these point sets are actually related within Euclidean space. This is the principal difference between the dynamic IC and the cellular IC introduced

⁶For instance, if we create N balls bouncing off M different surfaces, we do not have to define the behaviour of every ball separately taking into account the surface it is bouncing off. Instead we define the behaviour of one ball having an input parameter used to specify the collision object.

in (Kartasheva *et al.*, 2008). This is a compromise that allows us to simplify the actual description of IC models omitting certain details, which might be required for a valid definition of an IC from a theoretical point of view.

The relations within the dynamic ICs framework do not necessarily provide a complete description of the corresponding relations between the geometric point sets. Their set only includes those pairs of cells (i.e. those relations) that are explicitly evaluated during the dynamic process which is defined as the application specifics dictate. The same is true regarding other types of relations between point sets. Thus, some containment relations such as the “to contain” and the “to be contained” relations, as well as dependency relations of different kinds, do not need to be explicitly defined unless it is necessary. Note that if a pair of cells belongs to the boundary relation, it cannot belong to any containment relation at the same time. However, any pair of cells constituting both the boundary and the containment relations can also be related by dependency relations.

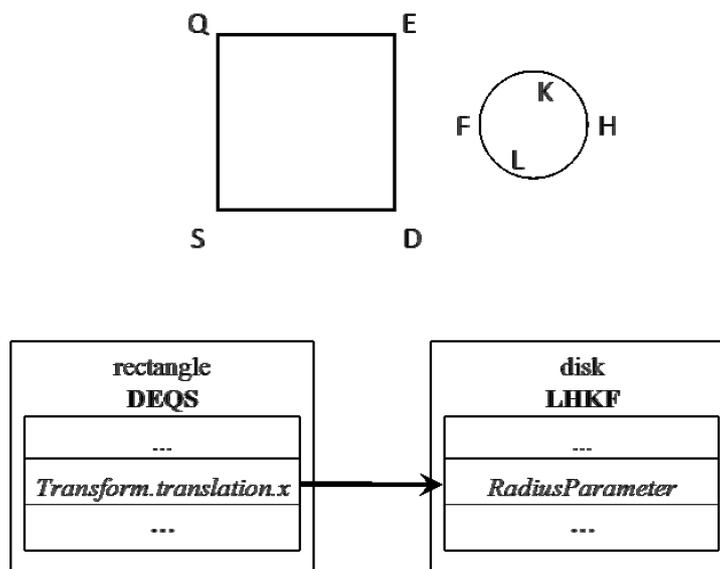


Figure 24: *The dependency relation between the parameters of two cells.*

We will refer to the example shown in figure 16 again. This time the example is extended through the introduction of a new dependency relation (fig. 24). A dependency relation is established between the translational component of rectangle DEQS and the radius of the disk LHKF. This results in the modification of the radius of the disk LHKF whenever the rectangle DEQS

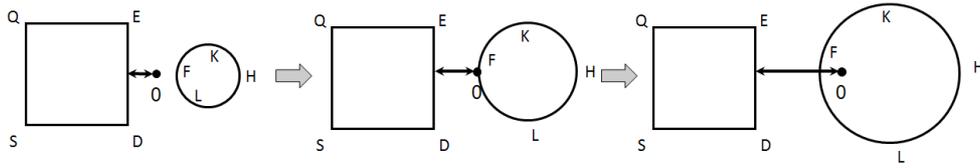


Figure 25: The modification of the radius depends on the translation of the rectangle.

is moved along the X-axis (fig. 25). In this case the cell LHKF is implicitly dependent on the cell DEQS. This is a simple example of a dependency relation. In the general case, the evaluation of a dependent parameter can be defined by a complex evaluation procedure.

3.6 Dynamic IC operations

The IC framework described in (Kartasheva *et al.*, 2008) allows us to define set-theoretic operations on cells. This is an important feature allowing us to construct composite geometric objects from a set of existing cells. The set-theoretic operations include the union, intersection and trimming of cells. In fact, we can also use a subset of the operations available for B-Reps (a set of deformations) and for FReps (set-theoretic blending and bounded blending). Theoretically the IC framework supports application of these operations for cells of an arbitrary representation, though, currently, these additional operations can only be applied to a set of cells of the same type. It is possible to perform an approximate conversion of a cell to an FRep or a BRep in order to apply the aforementioned operations between the cells of the same type. Exact conversion between the available representations is the subject of further research.

A new cell resulting from the application of an operation has an implicitly defined dependency relation with the cells that were used as operands (see section 3.5). For an n -ary operation the established dependency relations are defined as follows:

$$Rd_x^i = \bigcup_{j \in I_m^{i,x}} (C_j, C_i) \Big|_{p_{ij}^{Rdx}}$$

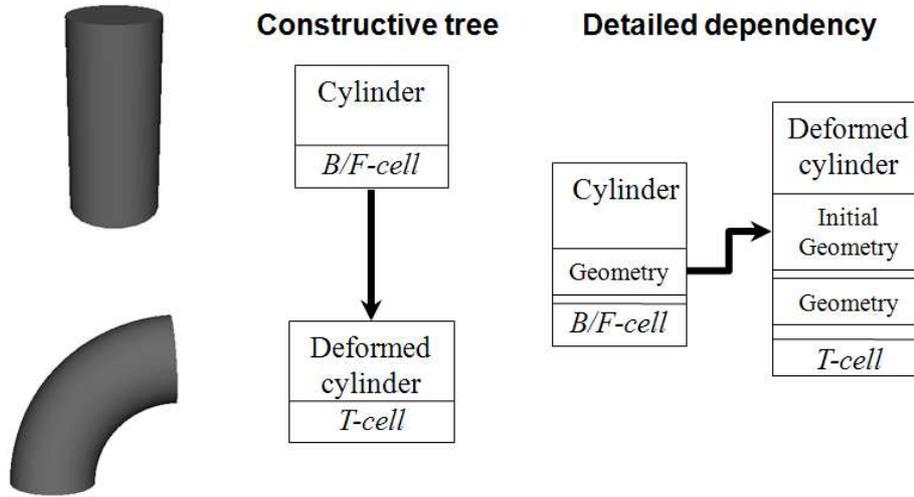


Figure 26: Example definition of a simple deformation.

where $p_{ij}^{Rd_x}$ is the priority of a dependency relation between the cell/attribute C_i and C_j , $I_m^{i,x}$ is the set of indices of the master cells (n operands of the operation in this case), i.e. C_i is a cell/attribute containing the operation that depends on the operand cells/attributes C_j . The application of an operation results in the modification of the components of C_i (including its geometry, attributes, parameters, etc.). Rd_x^i can be any type of dependency described in section 3.5, which means that operations can be applied to cells, attributes or combinations of these. The definition of operands through dependencies provides the user with the flexibility of the automatic tracking of changes in the model. Whenever the state of an operand cell is modified, the cell/attribute with the operations will be automatically updated. This allows us to define time-dependent CSG operations, blending operations, complex shape-driven deformations (Schmitt *et al.*, 2003) as well as various types of operations for attributes.

Fig. 26 depicts this idea. Here the initial geometry is defined by the cell “Cylinder”, which can be defined by either a BRep, an FRep or any alternative suitable representation. The geometric dependency relations established between the cells allow us to define a deformation which generates its geometry based on the provided input geometry (the “Initial Geometry” in this case). This allows us to store the constructive tree used to define compound objects.

It is important to note, that operations can be time-dependent in different ways:

1. The parameters of the operation remain constant but its operands are time-dependent:

$$\forall p_j(t) \in P_c^i(t) \Rightarrow p_j(t) = \text{const}$$

$$\exists t_1, t_2 : H_k^{qk}(t_1) \neq H_k^{qk}(t_2); t_1 \neq t_2$$

or

$$\exists t_1, t_2 : S_{\Lambda_k}(X, t_1) \neq S_{\Lambda_k}(X, t_2); t_1 \neq t_2$$

For instance, if we blend two moving objects the resulting shape will also be changing over time, even when the blending parameters remain unchanged (see fig. 27).

2. The parameters of the operation are changed over time, while its operands are not time-dependent:

$$\exists t_1, t_2 : P_c^i(t_1) \neq P_c^i(t_2); t_1 \neq t_2$$

$$\forall t_1, t_2 : H_k^{qk}(t_1) = H_k^{qk}(t_2)$$

or

$$\forall t_1, t_2 : S_{\Lambda_k}(X, t_1) = S_{\Lambda_k}(X, t_2)$$

In fig. 28 we can see that both the ball and the cube do not move. But blending parameters are increased over time which results in the modification of the resulting shape.

3. The parameters of the operation together with its operands are time-dependent:

$$\exists t_1, t_2 : P_c^i(t_1) \neq P_c^i(t_2); t_1 \neq t_2$$

$$\exists t_1, t_2 : H_k^{qk}(t_1) \neq H_k^{qk}(t_2); t_1 \neq t_2$$

or

$$\exists t_1, t_2 : S_{\Lambda_k}(X, t_1) \neq S_{\Lambda_k}(X, t_2); t_1 \neq t_2$$

Figure 29 demonstrates a model where two blended cells are moving in opposite directions while at the same time their blending parameters are adjusted.

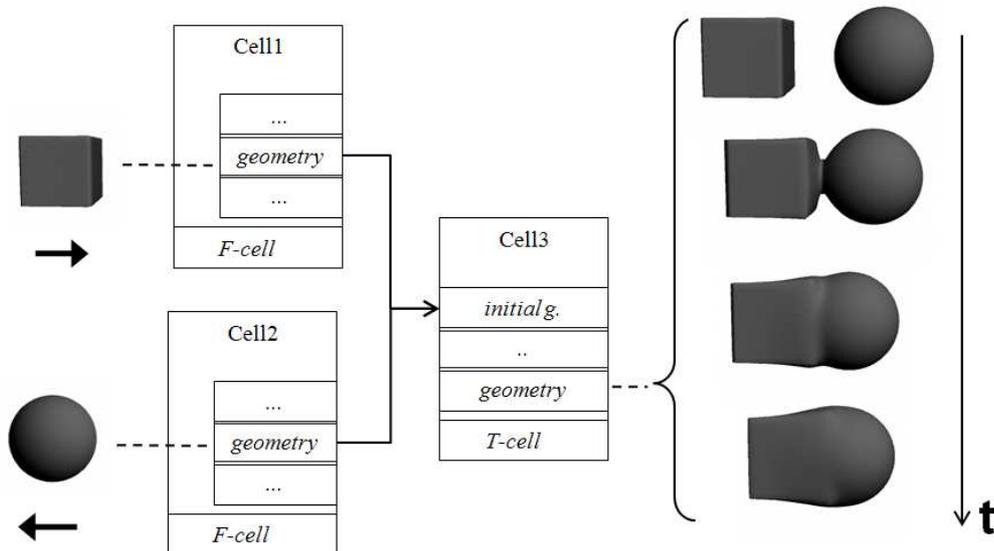


Figure 27: Example definition of an operation with constant parameters.

Certain dependency relations can be defined as persistent, i.e. they cannot be removed or modified. This type of relations appears to be useful for compound objects that set certain constraints on the modification of the resulting complex object. For instance, in the example shown in figure 27, if we remove the cell “Cell1” or remove the dependency relation, the T-cell “Cell3” would become invalid. This means that the shape of “Cell3” cannot be defined if no input geometry is provided to perform the evaluation.

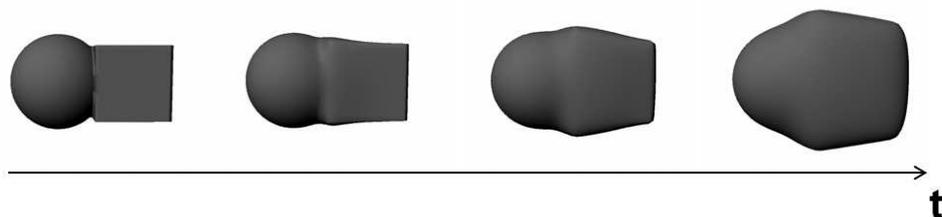


Figure 28: Example definition of an operation with time-dependent parameters.

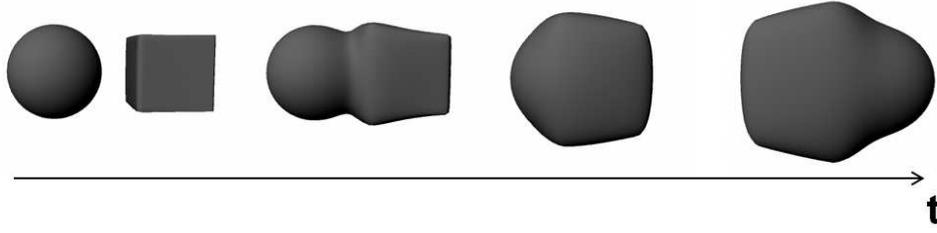


Figure 29: Example definition of an operation with time-dependent parameters and time-dependent operands.

3.7 Dynamic IC states and their components.

In this subsection we introduce a number of new terms that are required for the description of time-dependent states of IC entities.

We introduce the notion of the state of the dynamic IC cell. For an i -th IC cell we distinguish between the parametric state $S_i^p(t_j)$ and the structural state $S_i^s(t_j)$ of a cell:

$$S_i^p(t_j) = P_c^i(t_j)$$

$$S_i^s(t_j) = \{g_i^{q_i}(t_j), L_i\}$$

where $S_i^p(t_j)$ is a parametric state reflecting values of a set of parameters P_c^i of the i -th cell (defined in section 3.2) at the moment of time t_j and $S_i^s(t_j)$ is a structural state reflecting the modifications of the point set enclosed by the i -th cell (defined in section 3.2) together with its lifetime. The lifetime is included in the structural state as the point set enclosed by a cell outside of its lifetime is empty.

The state of the cell is then defined as a union of the two defined states:

$$S_{ji} = S_i^p(t_j) \cup S_i^s(t_j)$$

The change of a structural state would commonly result in a new parametric state of the cell as well. Transitions between the structural states can help distinguish between the changes of the geometry and the topology of the shape of the cell. Note that for the general definition of the geometry of any dynamic cell we can use functions of different types. For example, we can combine discrete shape functions with continuous motions or with deforma-

tions and so on.

In order to define a method for the definition of transitions between the available states we introduce a set of other terms. First, we introduce the concept of events. The occurrence of an event, normally but not necessarily, in a non-decreasing order of time, means that upon meeting certain conditions the behaviour of a model changes. The type of the event and a set of optional parameters are used to determine the condition :

$$E^i = \left\{ U^i, \bigcup_j^{N^i} P_j^i, T_i, p^i \right\}$$

where U^i is the name of the event, $\bigcup_j^{N^i} P_j^i$ is the set of parameters of the event, T_i is the time-span of the occurrence of an event and p^i is the priority of the event. Events with higher priorities are handled before events with lower priorities. The time-span of a predetermined event can be defined explicitly, otherwise it is set to the time of the actual existence of the event, if this period of time is not known in advance. Events can be created and processed by the “interested” cells (see the notion of reaction below). The predefined events include a change of time, the initialisation and the termination of an entity. User-defined events allow us to describe a set of events meaningful within a particular model, thus reflecting the consequence of modifications of the model state. Events may lead to both the modification of parametric and structural states of the entities.

In the example shown in figure 17 an event could occur when the disk first touches the perimeter of the rectangle or when the distance between the centres of the two objects falls below a certain value.

Each cell in the IC has a set of reactions associated with it. A reaction to an event is a mapping resulting in the transition of a cell from one state to another, which may produce a new set of events:

$$\Delta_i = \left\{ \bigcup_{E_j \in \mathbf{E}^i} \delta_{E_j}^i(S_j^i(t), t, P_j, \tilde{\mathbf{E}}^j) \right\}; \delta_{E_j}^i(S_j^i(t), t, P_j) : S_U \times T \rightarrow S_U$$

where $\delta_{E_j}^i$ is one of the “reaction” functions to the event E_j , \mathbf{E}^i is the set of events which the i -th cell reacts to, P_j is the priority of the reaction $\delta_{E_j}^i$, S_U is the set of states of the i -th cell and $\tilde{\mathbf{E}}^j$ is a set of events generated as a result of reaction $\delta_{E_j}^i$. A reaction can be defined using global time, local time and all the parameters of the cell. A set of new events $\tilde{\mathbf{E}}^j$ may be generated within the reaction. In addition to dependency relations this allows the IC entities to interact with each other within the model and to initiate reactions of other entities within the IC model.

Reactions are issued in response to events occurring in the system. The most common reaction is a reaction to the passage of time, initialisation and destruction events. Another predefined type of event is the request for an explicit modification of time (e.g., return to an earlier instance in time). Such an event can be used in order to adjust the time-step of the model, which would be required in order to retrieve the correct results of a physical simulation. The priorities of reactions provide a simple device for modifying the order of the reaction evaluation of the cells within a certain interval of time.

The motion of the disk shown in figure 17 is defined as a reaction to the passage of time, i.e. the position of the cell is modified at every instance of modification of global time.

Finally, a dynamic cell is defined as a composition of its initial state, its set of states and its set of reactions:

$$C_i = \{S_0^i, S_U^i, \Delta_i\}$$

Figure 30 illustrates the state transition graph of the states within the dynamic cell. Reactions of the cell are issued in response to external events. New events may be generated within the reactions in order to initiate further interaction with the model.

Any modification of global time results in the occurrence of an event. Reactions to this event are issued in the subset of the “interested” cells. This could result in global transitions between the states of the IC. Generally, time is modified automatically according to the specific requirements of the model. It is important to note that cells can request modifications of time in order to

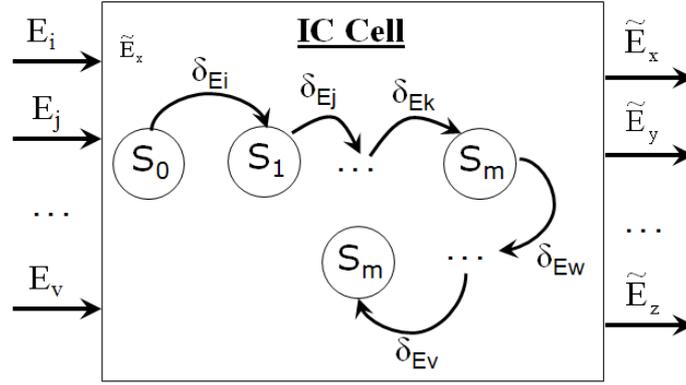


Figure 30: Transitions between the states of the cell through reactions issued in response to external events and a set of generated events.

evaluate their state correctly according to an active behaviour (e.g. in a physical simulation where the time step may need to be adjusted).

Analogously to the state of the geometric cell, we introduce the notion of the parametric state of a dynamic attribute at the time moment t_j :

$$S_i^{\Lambda p}(t_j) = P_c^\Lambda(t_j)$$

which is defined by the values of attribute parameters P_c^Λ , defined in section 3.3, at the time instant t_j . The custom parameters $P_c^\Lambda(t)$ can be used to modify certain properties of the attribute mapping or for other purposes. The structural state of the attribute over the i -th time span T_i is defined by the mapping $A^\Lambda(X, t)$, defined in section 3.3, and by the life span L_i of the attribute:

$$S_i^{\Lambda s}(t_j) = \{A^\Lambda(X, t_j), T_i, L_i\}$$

The state of the attribute at the time instant t_j is defined as follows:

$$S_i^\Lambda(t_j) = S_i^{\Lambda p}(t_j) \cup S_i^{\Lambda s}(t_j)$$

In a way similar to geometric cells, attribute entities can have their own reactions and a number of states (see fig. 30). Finally, a dynamic attribute is defined as a composition of its initial state, a set of states and a set of reactions:

$$H_i^\Lambda = \{S_0^\Lambda(t_j), S_U^\Lambda(t_j), \Delta_i\}$$

Associations of attributes and cells (attribute relations) are defined the same way as in the static case with the addition of priorities p^{RS} :

$$RS_{\Lambda} = \bigcup_{p=0}^3 RS_{\Lambda}^p \Big|_{p^{RS}}; RS_{\Lambda}^p \subset G^p \times S_{\Lambda}$$

$$RS_{\Lambda} \subset Rd_{ga}$$

where G^p is a set of p -dimensional cells. Thus, if $(g_i^p, S_{\Lambda_j}) \Big|_{\max p_{ij}^{RS}} \in RS_{\Lambda}^p$, the value of the attribute Λ at any point is defined as $S_{\Lambda_j}(X, t)$. The priorities of the attributes are used to resolve ambiguities, when certain point sets belong to a number of subsets of the geometry of the dynamic cells. In the case where more than one point set encloses the space, the attribute mapping with the highest priority value is chosen in order to evaluate the attribute value. Attribute relations can be thought of as a subset of dependency relations between the geometry and the attributes introduced in section 3.5.

3.8 Dynamic IC instances

In the previous sections we have proposed a set of extensions for the cells, attributes and relations, which constitute the basic building blocks used to define an IC.

Similarly to dynamic cells and attributes the entire dynamic IC model can be characterised by a set of states. ‘‘Structural states’’ within the IC model have a different meaning. If two states of a dynamic object are described by the same IC, then they are considered to have the same ‘‘structural state’’ despite any changes in the relative dispositions of the cells constituting the IC. The characteristic feature of a new structural state is the change in the description of the abstract IC relations. So it is possible, for instance, that some point sets of a pair of cells intersect in Euclidean space but the corresponding pair of the cells is not present in the sets of the boundary or containment relations, as those may not have been defined for that particular dynamic model. If we add this pair of cells into some abstract relation, then we effectively get a new structural state of the model without changing its point sets.

To reflect the structural changes of an IC based model over time (i.e. either by the establishment of new relations or by the removal of existing relations, or by the creation or destruction of some cells) we introduce the notion of an IC instance. Note that certain parametric modifications of the state can occur within an IC instance, but the structural state always remains unchanged within the life-span of an IC instance. The IC instance becomes invalid when a model undergoes such a structural change that it is no longer accurately described by the set of cells, attributes and relations used in the initial description of the IC instance. The validity of an IC instance is defined by a time-dependent predicate:

$$F^{I_j}(t^{I_j}(t)) = true \Rightarrow I_j(t^{I_j}(t)) \text{ is valid}$$

where I_j is a j -th instance of an IC, $t^{I_j}(t)$ is a transfer function used to retrieve the local time of the instance I_j ⁷ and F^{I_j} is a Boolean predicate associated with the IC instance. Every IC instance has its own local time starting at the instance in time when the IC instance became valid. Once an IC instance is found to be invalid, a transition to the next IC instance is performed (see fig. 31). The predicate of an IC instance implicitly has its own life-span (in a way similar to a cell). Only one IC instance can be valid at any given instance in time:

$$\forall i, j (j \neq i) : T^{I_i} \cap T^{I_j} = \emptyset$$

In section 3.7 we have provided a description of a reaction associated with a cell. Every instance has a set of reaction functions associated with it. These functions allow us to define the dynamic activities of an instance, to provide high-level reactions to external events and to generate new events (similar to those of the cells).

Note that the reactions of the cells and attributes can be customised within an IC instance. Thus if required, a new set of reactions can be defined for any cell or attribute present in the IC instance:

$$\exists C_m \in \bigcup_{i=1}^{N_j} C_i^{I_j} : C_m \in \bigcup_{i=1}^{N_k} C_i^{I_k} \Rightarrow C_m^{I_j} \cdot \Delta_m \neq C_m^{I_k} \cdot \Delta_m; j \neq k$$

⁷A transfer function is used in order to take into account the time span of an IC instance and the rate at which its local time is modified.

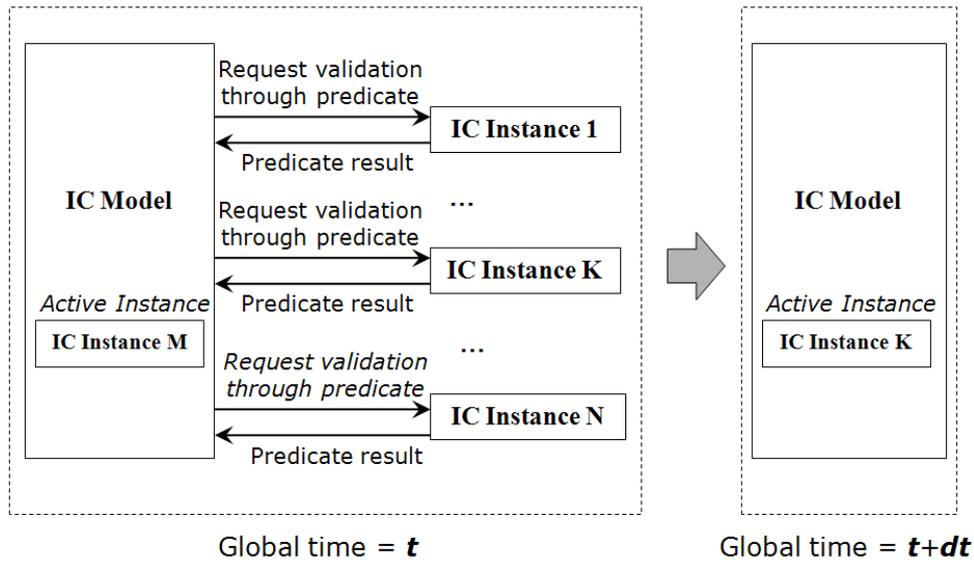


Figure 31: Transitions between the IC instances of the IC-based model.

where Δ_m is a reaction of the cell C_m . This allows us to alter the behaviour of any cell or attribute depending on the current state of the model.

Every IC instance can have its own set of parameters ($P_c^{I_j}$) together with a set of parameters for its components. Both these sets of parameters can be used for the manipulation of the model, as well as for the tracking of the state of the model over time. Thus, we provide a way to define a set of state parameters “linked” either to a set of instance parameters or to a set of parameters of any cell present in the IC instance:

$$P_s^{I_j} = P_{s,I_j}^{GA} \cup \bigcup_k^{N_j^{Is}} P_{c,k}^{I_j}$$

An example of such parameters could be the velocity of some geometric cell that is important in the context of the model or the distance between the shapes of two cells (introduced as a parameter of an IC instance).

Finally, an IC instance is defined by a set of cells and a set of relations present in it, a set of attributes and the attribute relations for the cells, a set of internal parameters of the instance, its life span, as well as its predicate

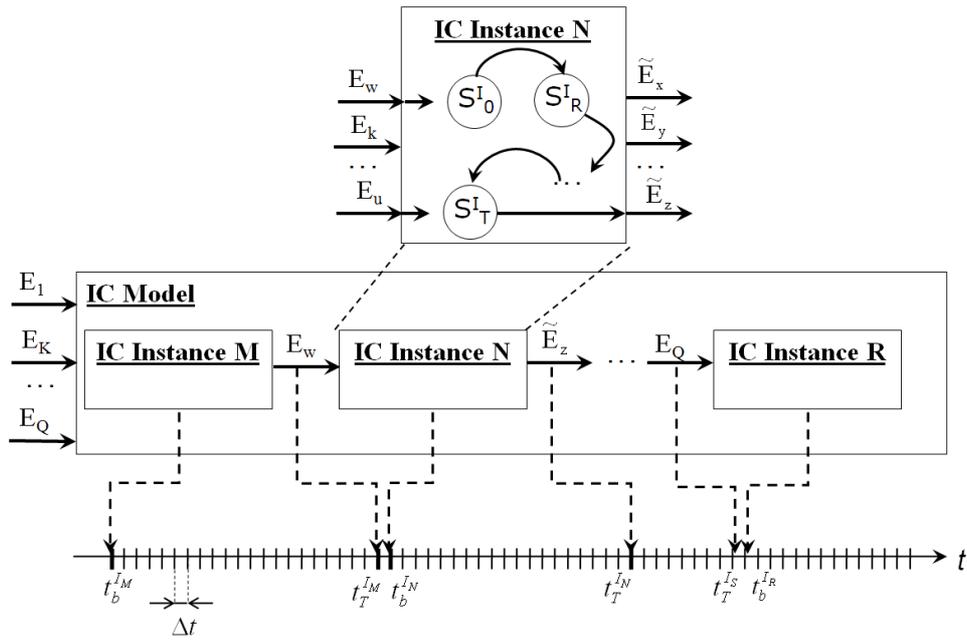


Figure 32: Dynamics of transitions between the structural states of the model over time.

function and reactions (P_{s,I_j}^{GA}):

$$I_j = \left\{ \bigcup_{i=1}^N C_i^{I_j}, Rb^{I_j}, Rc^{I_j}, Rd^{I_j}, \bigcup_{k=1}^M H_{\Delta k}^{I_j}, Rs_{\Delta}^{I_j}, P_c^{I_j}, L^{I_j}, F^{I_j}, \Delta^{I_j} \right\}$$

There is no need for an explicit definition of the structural state of the IC instance. The IC instance embodies the structural state as a collection of components of the dynamic IC-based model. Any change of the state of the intrinsic components of the IC instance results in a transition between the parametric states of the IC instance itself (see fig. 32). The transition between the parametric states is performed through reactions of entities within the current IC instance and through the reactions of the dynamic IC instance itself. Transitions between the IC instances, which are equivalent to the transitions between structural states, occur only when a set of components constituting the model at a specific moment of time is modified (see figure 32). Such a modification is indicated by the value of the predicate of the IC instance, which allows us to determine if a transition to a new structural state is required.

3.9 The IC-based model

In this section we introduce a set of terms used for the description of the entire model.

Using the definition of dynamic point sets we formulate the definition of a dynamic IC-based model as a collection of dynamic cells, attributes and relations between these:

$$Q(t) = \left\{ \bigcup_{r=0}^n \bigcup_{i=0}^{Nr(t)} g_i^r(t, P_c^i(t)), \bigcup_{k=1}^M H_k^\Lambda(t, P_c^i(t)), R_b, R_c, R_d \right\}$$

All the cells, attributes, relations between these as well as with the number of these relations can be time dependent. We do not impose any restrictions on the mutual dispositions of the geometric cells.

Another important aspect of the model definition is the parameterisation of the model. Every cell or attribute in an IC instance has a set of parameters that may have some higher level semantic meaning within the model. We collect a subset of these parameters into a special set:

$$P_{s,I_j}^{G\Lambda} = \bigcup_k \bigcup_i^{N_j^G N_{k,j}^{PG}} g_k^{I_j} \cdot P_{c,i} \cup \bigcup_k \bigcup_i^{N_j^\Lambda N_{k,j}^{PA}} \Lambda_k^{I_j} \cdot P_{c,i}$$

where $g_k^{I_j}$ and $\Lambda_k^{I_j}$ are the k -th cell and attribute belonging to the IC instance I_j , $N_{k,j}^{PG}$ is the number of the meaningful model state parameters of the cell $g_k^{I_j}$ belonging to the instance I_j and $N_{k,j}^{PA}$ is the number of meaningful model state parameters of the attribute $\Lambda_k^{I_j}$ belonging to the instance I_j .

An abstract dynamic IC is defined as the set of its IC instances. Each instance includes a set of the cells along with the established relations between these. As the IC is a discrete entity (i.e. the set of its cells is discrete as well as its relations are subsets of the Cartesian products of discrete sets of cells), then its dependency on time can be considered as a change of its consequent states which are associated with the instances of the IC. Two approaches could be used here.

The first approach requires the introduction of “the IC template” which includes the union of all the cells and relations belonging to all the static IC instances. The set of time-dependent predicates associated with the cells, its attributes and their relations are also necessary. Thus to define the IC instance at any given time, one needs to evaluate all the predicates and then to remove all the cells and relations which are not “valid”. This results in the IC instance for that time instant. The main advantage of this approach is the uniform description of all the states. Its obvious drawback is the redundancy of the representation. Additionally, it is difficult to define the validity of all the possible states in terms of the restrictions imposed onto the relations between the cells, especially when the complete set of the states is not known in advance.

The second approach assumes the existence of a succession of IC instances. All the ICs from this well-ordered finite set include the relations which are common to all of them. To define the dynamic IC one only needs to set the equivalence relations between the cells of those IC instances that are adjacent in terms of time. This will allow for comparisons between relations in terms of cell pair numbers and compositions. The advantages of such an approach are twofold:

- all the states are known in advance and their validity can easily be checked;
- there is no need for redundant cells in each static IC.

The drawbacks include the necessity to store the descriptions of all the ICs and the impracticality of defining the intermediate states on the fly.

It makes more sense to combine both outlined approaches. We could define the IC template with some predicates and each state (“IC instance”) could be defined on the basis of this template with the option to add the cells, attributes and relations which were not present in the template. Certain cells, attributes and relations could also be removed from the template, in order to define a new IC instance. One could also introduce the predicates associated with each IC instance. This would allow for an easier evaluation of the validity of the IC instance as a whole.

Let TI be a template of the IC that is used to create various instances of that IC. The IC template can be seen as the IC instance prototype. New instances are created through the addition or removal of some components of the IC template (e.g. new cells or relations). This simplifies the process of model definition, as in most cases the model undergoes a set of minor modifications over time. Hence, most of the entities are present in many IC instances, which means that we do not need to provide an exhaustive definition of every instance from scratch. Instead, we add or remove some components using the template TI :

$$\exists j : I_j = TI \setminus \left\{ \bigcup_{i=1}^{N_l} C_i^{TI}, Rb^{TI}, Rc^{TI}, Rd^{TI}, \bigcup_{k=1}^{M_l} H_{\Lambda k}^{TI}, R_{s_{\Lambda}}^{TI}, P_c^{TI}, F^{TI}, \Delta^{TI}, \right\} \\ \cup \left\{ \bigcup_{i=1}^{N_m} C'_i, Rb', Rc', Rd', \bigcup_{k=1}^{M_m} H'_{\Lambda k}, R'_{s_{\Lambda}}, P'_c, F', \Delta' \right\}$$

Additionally, we need to provide a definition of an animation entity. This entity can be used to define a key-frame-based animation for any parameter present in the model (i.e.: it is “attached” to the parameter):

$$A_i(t) = \left\{ \bigcup_{j=1}^{N_F^i} (t_j^i, v_j^i), \bigcup_{k=1}^{N_E^i} (t_{Ej}^i, v_{Ej}^i), T^i, F_{A_i}^{in}(t), F_{A_i}^{out}(t) \right\}$$

where N_F^i is the number of key-frames of the animation $A_i(t)$, (t_j^i, v_j^i) is the set of tuples of time moments and parameter values at these time moments, (t_{Ej}^i, v_{Ej}^i) is the set of possible additional values required for a specific type of interpolation, T^i is the time-span of the animation (i.e. the interval including the time of the first and last key-frames), $F_{A_i}^{in}$ is the function performing the evaluation of the parameter values within the time-span of the animation (e.g. this can be one of the predefined polynomials or a custom evaluation procedure) and $F_{A_i}^{out}$ is the function performing the evaluation of the parameter values outside the time-span of the animation (e.g. constant, linear, custom, etc).

Finally, the entire model is defined by a set of IC instances valid over different periods of time, a set of global parameters, a set of custom events

and a set of animations used in the model:

$$M(t) = \left\{ \left\{ \bigcup_{j=1}^L I_j(t^{I_j}(t)); F^{I_j}(t^{I_j}(t)) = true \right\}, \bigcup_{k=1}^M P_k^g(t), \bigcup_{u=1}^N \tilde{E}^u, \bigcup_{s=1}^Q A_s(t) \right\}$$

The description of a set of IC instances should be provided by the user of the system after the process of model decomposition. As was outlined in section 3.8, an IC instance embodies the structural state of the model. The parametric state of the model is defined through a set of dynamic cells and attributes together with their parameters and the global model parameters:

$$S_M^P(t_j) = \left\{ \bigcup_{i=1}^{N(t_j)} C_i^{I^{M(t_j)}}(t_j), \bigcup_{k=1}^{M(t_j)} H_{\Lambda k}^{I^{M(t_j)}}(t_j), P_c(t_j), P_c^g(t_j) \right\}$$

The parametric state of the IC model is simply a snapshot of the model at the moment of time t_j . Figure 32 illustrates transitions between the IC instances and transitions within the internal parametric states of these instances over the course of the modelling process. Figure 33 outlines various components of the model and the relations between them.

We refer to a simple motion illustrated in figure 17 once again. All the cells and relations initially present in the IC are outlined in figures 16 and 34. Both the set of cells and the set of relations are modified over time. For instance, when the disk touches the rectangle a new cell is introduced (fig. 35).

Observe that a new constructive tree cell F has been added at the intersection of the two initial cells. Also, two new cells and a set of new relations are added at a subsequent moment of time (fig. 36).

All these descriptions of the intermediate states of the model are represented using different instances. Each of the instances also has a predicate associated with it, allowing us to distinguish between the different states of the IC model over time.

The transitions between the IC instances analogously to the diagram in fig-

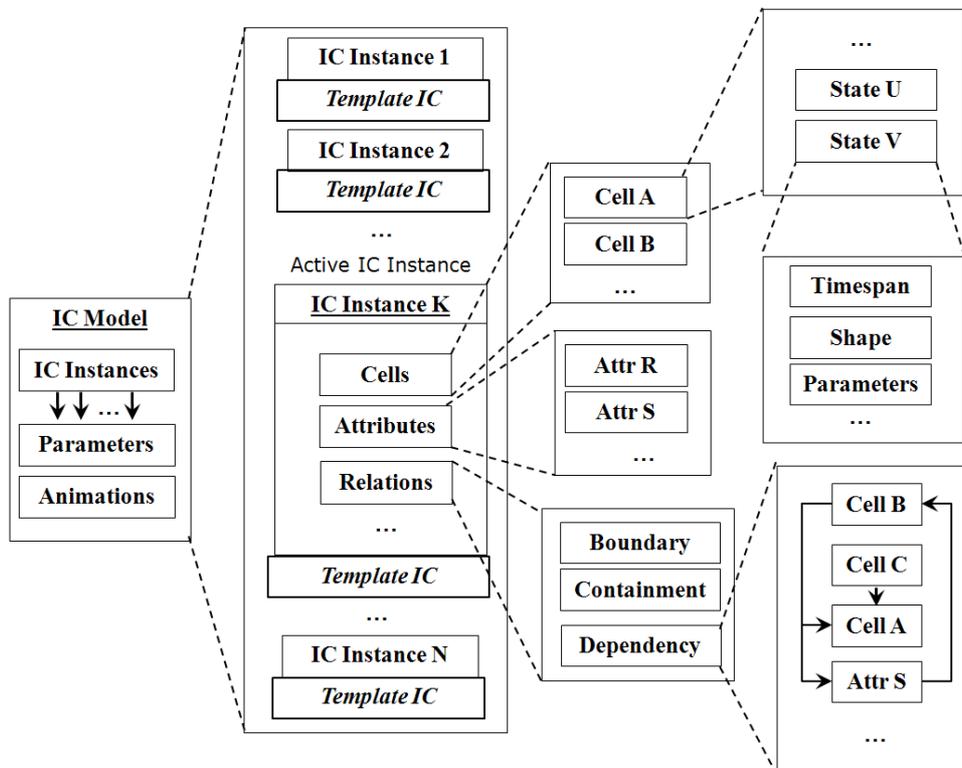


Figure 33: The structure of the dynamic IC model and its components.

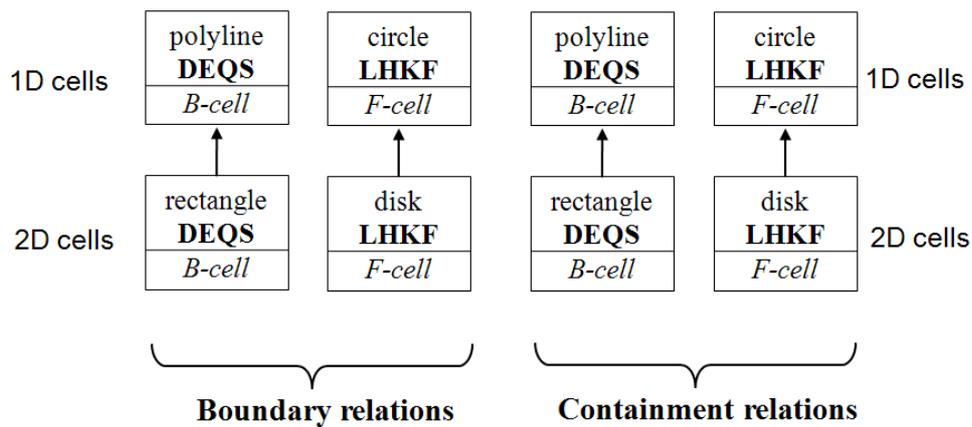


Figure 34: The topological relations of the IC.

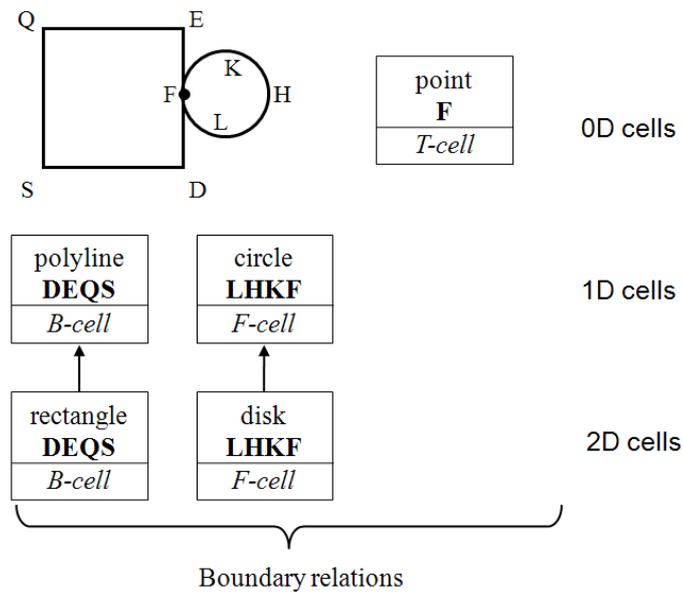


Figure 35: The set of cells at an intermediate modelling phase.

ure 32 are shown in figure 37. This illustration depicts the different structural states of the model at different time instants. We can see that these intermediate structural states of the dynamic model remain valid over different periods of time.

3.10 Conclusions

In this chapter we have introduced a new theoretical framework which can be used for the definition of mixed-dimensional time-dependent heterogeneous objects. This new framework is based on the previously available IC framework that was used for the definition of static hybrid models. The static IC framework served as the basis for the introduction of the new concepts and for the extension of the existing notions. We have consistently extended the definitions of IC cells and attributes in order to accommodate their new dynamic features. Each entity can now be characterised by its two-fold state, namely its structural and its parametric state. We have then introduced a set of new dependency relations which are crucial for complex dynamic models. The introduction of events and reactions of the IC entities to these events allows us to define complex dynamic models through event-driven dynamics.

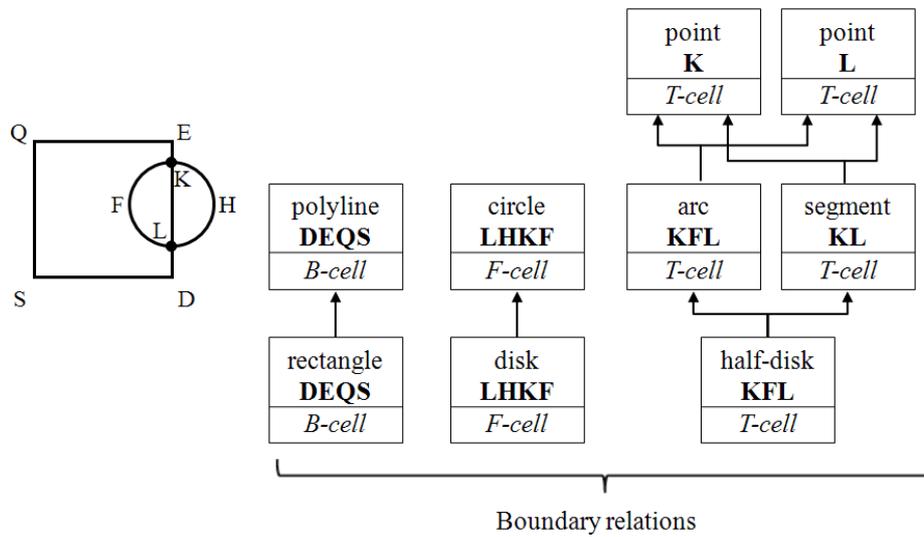


Figure 36: The set of cells at an intermediate modelling phase.

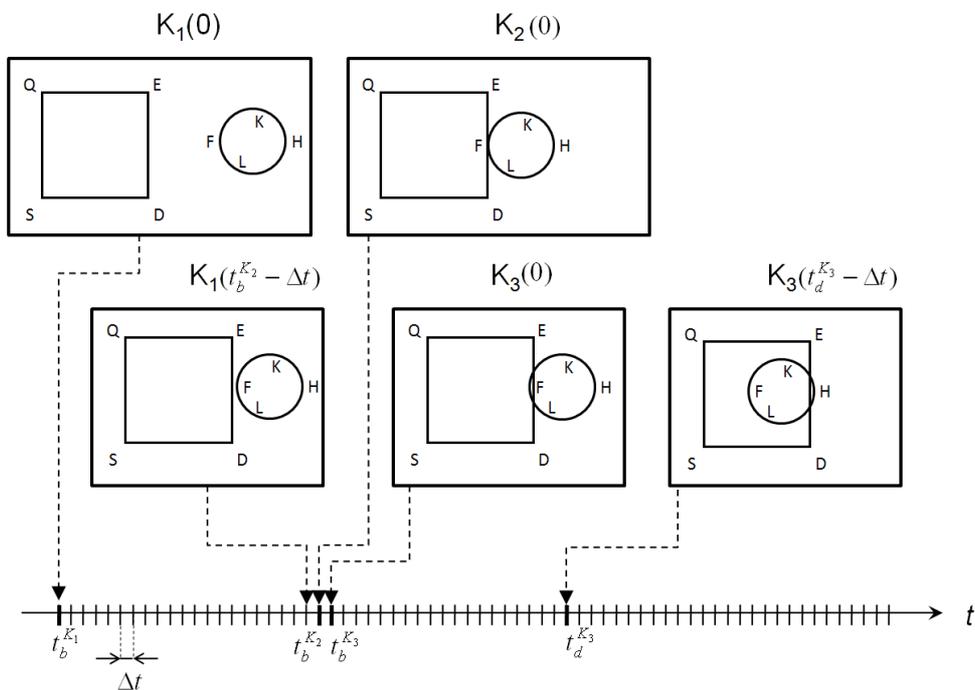


Figure 37: A set of instances reflecting the model at different time-spans.

The new set of features available in the dynamic IC framework allows us to combine the definition of heterogeneous objects together with their complex behaviour in different ways. Apart from the definition of the behaviours of individual objects we can also specify compound relations between these objects. Topological relations allow us to build and retrieve information about the mutual dispositions of dynamic objects over time. Dependency relations make it possible to define elaborate hybrid models in a modular fashion, building complex dependency graphs between entities present in the model.

In the next chapter we will provide a more detailed description of this framework, its internal structure and its implementation.

4 Dynamic Implicit Complexes Framework: Technical Aspects and Tools.

In this chapter we describe the system design of the framework introduced in the previous chapter. The provided description deals with a more detailed definition of a number of internal aspects of the framework. Here we provide descriptions of various components of the hybrid model and we describe the process of the model evaluation. We introduce a high-level notation which can be used for the definition of a time-dependent hybrid mode. We then focus on the part of the framework related to FReps. Finally, we discuss techniques that can help us improve performance of the IC framework evaluation.

4.1 Introduction

In the previous chapter we have introduced a new framework allowing us to deal with mixed-dimensional dynamic hybrid models whose structure and properties change over time. This theoretical framework makes it possible to incorporate the existing representations and to overcome some of their limitations. Our new dynamic framework allows us to combine a procedural definition of time-dependent model, based on event-driven dynamics, with the widely-used traditional keyframe-based approaches. However the theoretical description of the framework so far is not detailed enough to allow us to start building actual hybrid models. We need a practical system which can evaluate a mixed-dimensional hybrid model and can output the results given a valid description of a model. The definition of a model should rely on all the terms introduced in the theoretical description of the dynamic IC framework (see chapter 3). Further in this chapter we will provide a detailed description of the components of the framework and their appropriate counterparts in a practical system. After this we present a notation and introduce a new language for the definition of hybrid dynamic IC models. We will also describe the algorithms required for correct model evaluation together with important details required for the implementation of the proposed dynamic IC framework. The information provided should be sufficient to allow for

a full-featured implementation of the IC framework based on the theoretical groundwork provided in the previous chapter.

4.2 A description of the IC entities and their properties

In the previous chapter we have introduced the theoretical groundwork for the new dynamic IC framework. Our theoretical description should be sufficient for a basic understanding of the concepts behind hybrid modelling. The conceptual dynamic hybrid model can be defined using the set of available notions that we have introduced earlier. But we need a more practical way of describing the IC-based models.

Here we provide a description of all the entities present in the framework from the perspective of system design in an object-oriented manner (Booch, 2004). We map an entire set of notions introduced in sections 3.2-3.9 to a set of entities available to the user for the definition of a dynamic hybrid model. The high-level UML diagram shown in figure 38 allows us to map all the theoretical concepts, presented in the previous chapter, into a set of more concrete practical terms. A description of a hybrid model can be developed through the composition of the descriptions of entities such as events, cells, attributes and IC instances. Entities, in their turn, are defined through the values of their components. This makes the step-by-step iterative definition of a complex IC-based model possible in a natural manner. A more detailed specification of the entities and their components is presented in Appendix A.

In order to provide a valid description of the model, definitions of all its components need to be provided. Initially we describe all the cells of the IC independently. Then, for each instance of the IC, we add cells that only exist within this instance. All IC instances also have access to any cell described within the entire IC definition (see fig. 38). The state of certain T-cells needs to be validated by the instances, i.e. the T-Cells remain “invalid” unless their state has been updated by some evaluation procedure defined in the instance, which means that instead of an explicit definition of parameter values, the user can provide a specific description or an evaluation procedure allowing us to evaluate the actual parameter values.

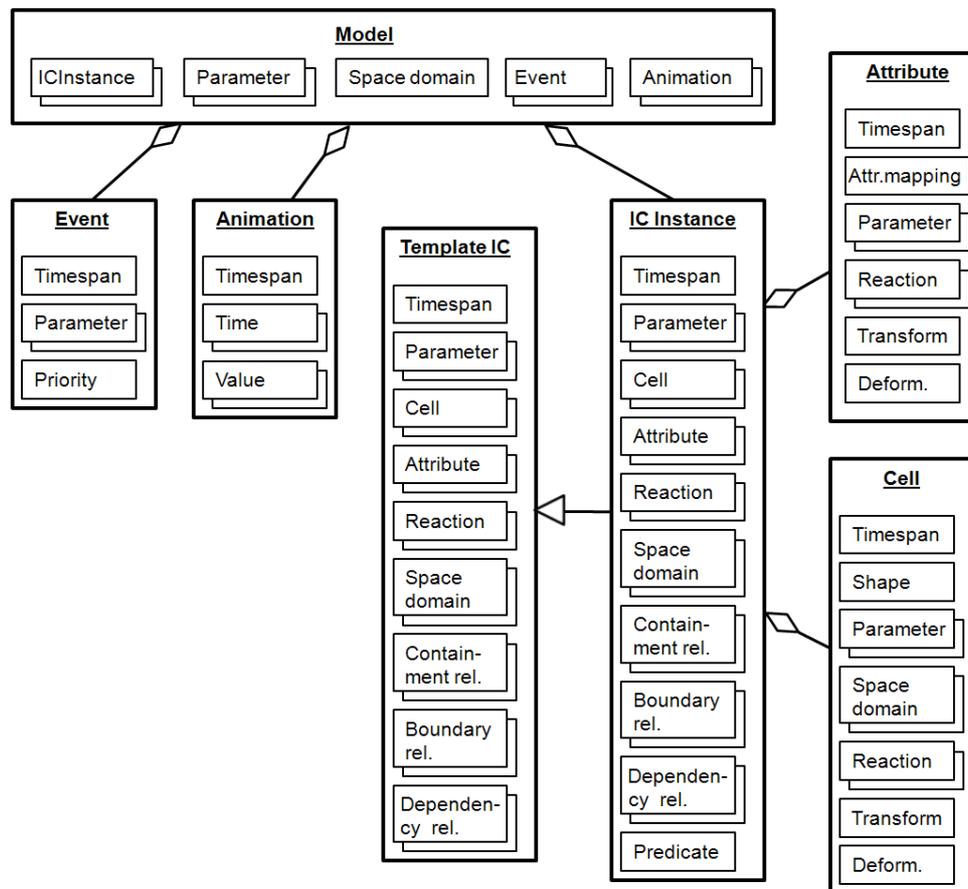


Figure 38: A high-level UML diagram of IC entities.

There are supplementary methods available for the user to determine any intersections/collisions between the objects. These are needed to simplify topological queries for dynamic objects - for instance, find out whether the objects are touching or intersecting each other.

The information presented in Appendix A in tables 2 and 3 can be used as an initial guidance for the implementation of the described dynamic IC framework.

The independent definition of the components of an IC-based model is not sufficient for a valid description of the model. The user needs to follow a specific methodology in order to map his mental image of the model to a practical model description, which can be used for evaluation. In the next section we describe the set of steps necessary for the definition of a dynamic hybrid model.

4.3 Methodology of model definition

There is no easy way to decompose a complex heterogeneous object or set of objects in a set of IC cells and attributes. There may exist a number of ways to achieve this. Nevertheless, here we provide a sequence of actions typically required for the definition of a dynamic IC model using the entities depicted in fig. 38. The steps commonly required for the definition of the model are as follows:

- Define a set of custom events having a specific meaning within the model (see section 3.7), if such events are required.
- Define the “independent” cells initially present in the IC template (see section 3.2). A list of IC cells can be retrieved after the decomposition of the conceptual model. Each cell reflects an entity in the original model with a set of properties. The list of these cells can be modified over the course of the iterative model definition. The properties of the cells and their optional reactions to a set of default or custom events should also be defined here.
- Define the “independent” attributes initially present in the IC template (see section 3.3). The list of these attributes can be modified over the course of the iterative model definition. The properties of the attributes and their optional reactions to a set of default or custom events should also be defined here.
- Define the IC template(s) that will be used as the basis for the definition of all the IC instances (see section 3.8). At this stage the cells and attributes defined previously as well as the relations between them should be defined. Additional events and some predefined cells and attributes from the library can be used at this stage. A set of relations between the cells and between the attributes can also be defined here. A detailed description of the IC template can greatly simplify the definition of all the IC instances, because new instances can be described through the introduction of minor modifications to an IC template definition.
- Define a set of IC instances over time. As we mentioned before (see section 3.8), IC instances reflect states of the model which are struc-

turally different from each other (in terms of the cells, their attributes or relations present in them as well as different behaviours of their components). The user also provides a description of the period of time over which the modelling is performed as well as the list of sampling parameters of the model. Each instance can include a set of IC template cells, attributes and relations. New cells, attributes and relations relevant only to the current instance are also introduced here. Moreover, the reaction, introduced in section 3.7, of any cell/attribute can be altered in order to modify the behaviour of any cell/attribute over the life-span of the IC instance. The relations between the attributes and the cells are also provided at this stage. Finally, the predicate providing the IC instance with validity information should also be defined here.

- Add a set of parameters specific to certain IC instances and global parameters reflecting the state of the model. These parameters can be used to gain a better understanding of the state of the model or to alter its behaviour if it changes on the fly, as was outlined in section 3.9.
- Define the set of the modelling parameters (i.e. the modelling time domain, the desired parameters of the model, the rendering parameters, the simulation step and others).

The dynamic IC framework engine can now start model evaluation using the provided model description. The result of the evaluation may include a set of parameters evaluated over time, the complete states of all the entities over the course of the modelling process or a sequence of the rendered frames with a visual representation of the model at consecutive instances in time.

4.3.1 The high-level notation for the definition of an IC model

In this section we present a brief description of the new high level notation used to describe a dynamic IC model.

Once a conceptual model has been designed according to the general methodology presented in the previous section, it needs to be transformed into a description which can be provided to the IC framework. This description can then be used to build the internal data structures and to set up the algorithms

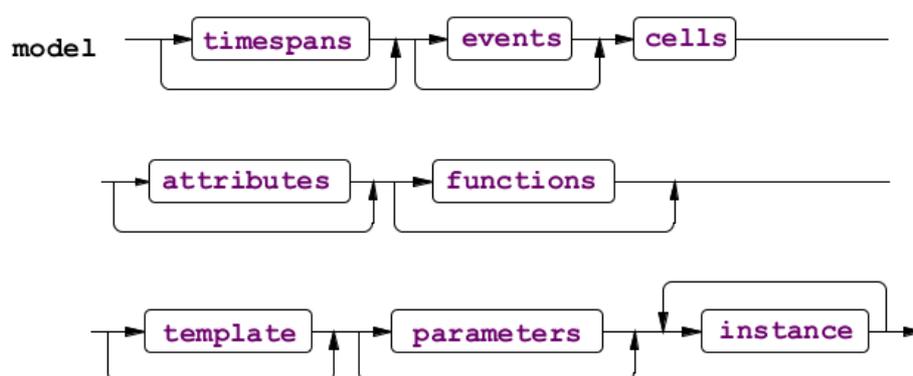


Figure 39: *The structure of the textual IC model definition.*

required for the evaluation of the hybrid model. Here we provide the most relevant sections of a symbolic model description using syntax diagrams.

According to the definition of the dynamic IC model, described in section 3.9, the textual description of the model consists of the description of custom events, dynamic cells and attributes, a template IC and a set of IC instances (see fig. 39). Additionally, custom time-spans and commonly used functions (described in section 4.3.2) can be defined for the ease of modelling.

Let us now provide syntax diagrams for the most important parts of the textual model description.

Custom events and their parameters are described first (see left of fig. 40). The names of the events are used later by the cells and the instances in order to provide reactions to these events. Additionally the time-spans of these events can be defined here, if the moments and intervals of their occurrence are known in advance.

After the events and time-spans the “persistent” cells⁸ present in a number of instances of the IC are described. The user can define all the components of the cells according to the definition in section 3.7 (see right of fig. 40). “Source” operator “<-” is used to indicate that the description of the cell is

⁸These are entities that exist in all or a number of instances of the IC-based models. Cells/attributes present in only one IC instance can be included within the description of an instance, as the references to these cells won’t be used in any other IC instance of the model. Otherwise, descriptions of the cells may become too complicated and overloaded with the details of the model relevant only at specific moments of time. In a way this is similar to a well known separation between global and local variables.

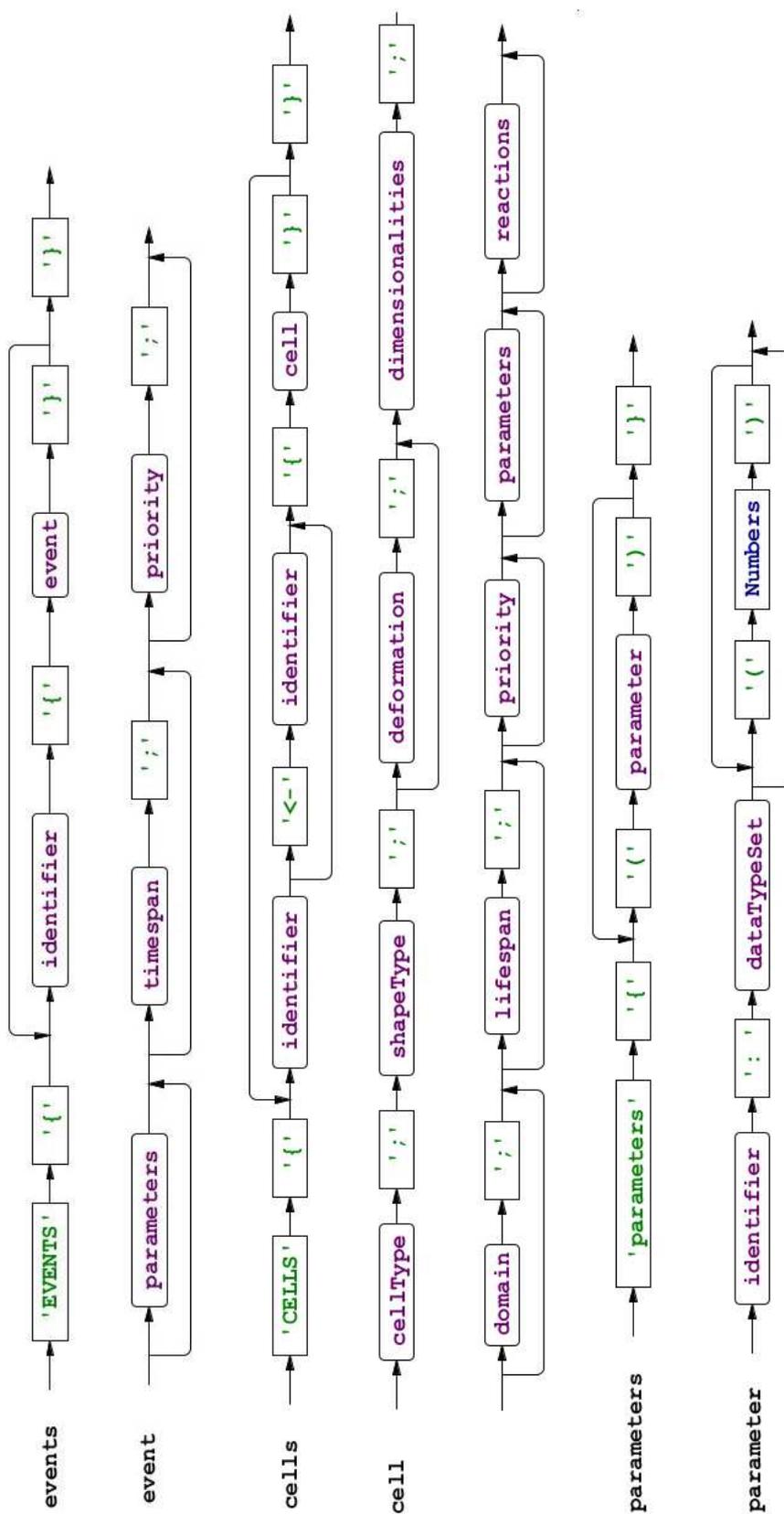


Figure 40: The syntax diagram for the definition of entities present in the model.

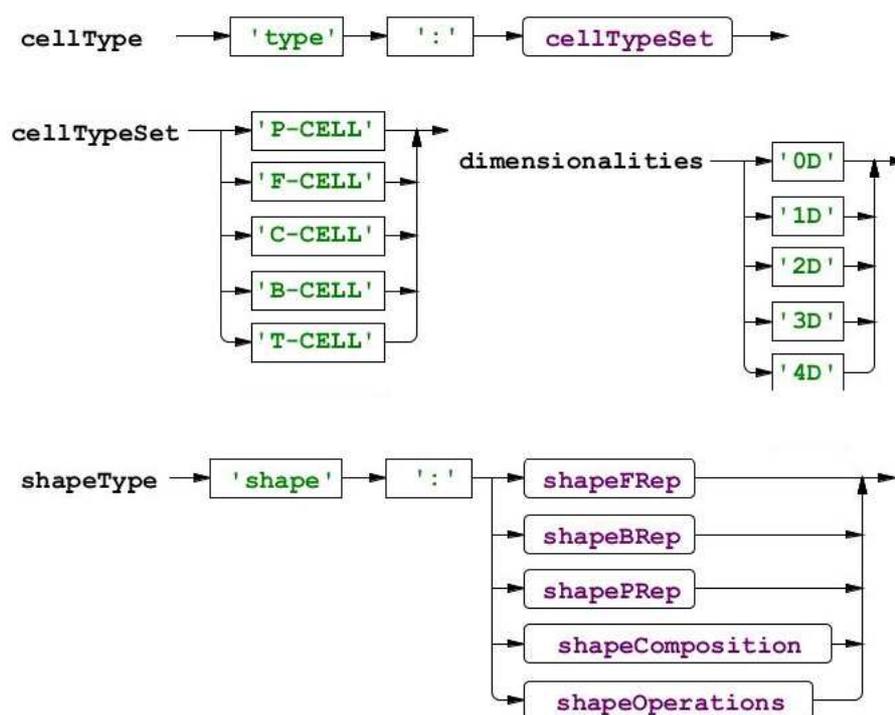


Figure 42: The syntax diagram for the definition of internal components of the cell.

based on the definition of another cell (i.e. undefined values of the properties will be equal to the values of the “sourced” cell). The syntax for the description of a set of important components of the cell is depicted in fig. 42. “Persistent” attributes are defined in a similar manner, according to the definitions provided in section 3.3. The reactions of all IC entities are defined in a similar fashion according to the definition in section 3.7. The syntax for the description of these reactions is provided in fig. 41. The `react` function allows the user to define reactions to custom events referenced by name. The `reactionParams` include information regarding both the global and local times at the time instant when a reaction is issued. The `update` the reaction is a predefined type of reaction to the modification of time, while `init` and `term` reactions are issued when an entity is created and terminated respectively.

Before all the IC instances are defined an optional definition of an IC template (see section 3.8) can be provided for the ease of further modelling. Fig. 43 illustrates the syntax used for the definition of the IC template and a set

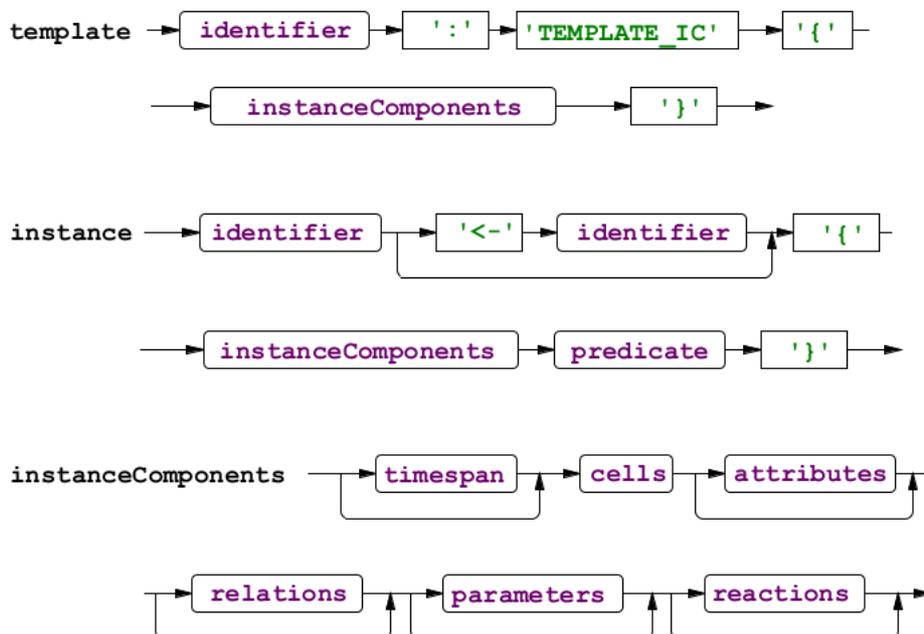


Figure 43: The syntax diagram for the definition of template ICs and IC instances.

of IC instances. In this case the “source” operator “<-” is used in order to indicate that a particular IC template should be used as a prototype of an IC instance (see section 3.9).

In the next section we will provide a brief overview of the language constructs which should help the reader gain a better understanding of the notation used for the definition of dynamic hybrid models presented later in the text.

4.3.2 IC model definition using the notation

Here we provide a high level overview of each IC component in the order that would be commonly used for the definition of a model. This is the same order as the one used in the previous chapter when introducing the general dynamic IC notation. This order is also reflected in syntax diagram of fig. 39. Here we use a pseudo Extended Backus-Naur Form (EBNF) notation to provide an illustrative description of the syntax.

Custom events and their parameters are described first (see diagram in fig. 40). The names of events are used later by the cells and the IC instances in order to provide reactions to these events. Additionally the time spans of these events can be defined here. For instance:

```
// custom time-spans used in the model
TIMESPANS {
  // span identifier and actual span parameters (default span is an
  // infinite interval with time scale = 1.0)
  spanName1 : TIMESPAN(...);
}
// description of events that can occur in a model
EVENTS {
  // list of events.
  EVENT_NAME { // name of the event is used to provide a reaction
    // set of parameters/properties describing this event (optional)
    parameters {
      paramName ::= TRANSFORM | VECTOR(...) | REAL(...) ...;
      :
    }
    [priority ::= REAL; ] // optional priority
  }
}
```

The “independent” cells described in the previous section are defined next. All the terms used for the definition of all the components of the cells are taken from section 3.2. Here we provide the syntax used for the definition of dynamic IC cells according to syntax diagrams depicted in figures 40 and 42:

```
CELLS { // list of cells starting with the name of the cell
  // "<->" operator means to make an initial copy of a cell
  cellName [ <- sourceCell ] {
    // representation
    type ::= P-CELL | F-CELL | C-CELL | B-CELL | T-CELL;
    // actual shape (can be a composition of existing cells)
    shape ::= MESH | FREP_TREE | PREP_PRIMITIVES |
      COMPOSITION_OF_CELLS | ...;
    dim ::= 0D | 1D | 2D | 3D | 4D; // // dimensionality
    // domain of a this cell (e.g. bounding box)
    domain ::= ( VECTOR_OF_VALUES; VECTOR_OF_VALUES );
    // optional life span parameter
    [ lifespan ::= ( REAL_VALUE; REAL_VALUE ) U
      ( REAL_VALUE; REAL_VALUE )... ];
    [ priority ::= REAL_VALUE ] // optional priority value
    // set of parameters/properties describing this cell
    parameters {
      // predefined parameters can be initialized too
      [
        translation(...);
        rotation(...);
        scale(...);
      ]
      // define new parameters and their initial values:
      paramName ::= VECTOR4(...) | VECTOR3(...) | REAL(...) ...;
      :
    }
  }
}
```

```

}
// optional description of a deformation (can be external)
[ deformation ::= ... ]
:
:

```

Apart from the description of the static properties of the cell, the default time-dependent reactions of the entities can also be described within the cell definition (see fig. 41). If a different reaction of the entity is provided within the description of an IC instance, this reaction is used instead of the default one.

```

REACTIONS { // reactions to events
  // default reactions (can be redefined in an IC instance).
  init( REAL globalT) {...}
  terminate( REAL globalT) {...}
  // the framework provides the current global and local time
  // within different time-spans for all reactions
  update( REAL globalT, REAL localT, REAL lifeT, REAL dt){...}
  // reactions to custom events.
  react(EVENT_NAME eventX, REAL globalT, REAL localT, REAL lifeT)
  {
    param = eventX.paramS;
    :
    :
  }
  :
}

```

Attributes are described in a similar fashion (according to the definitions in section 3.3). Unlike cells, attributes need to be associated with a mapping to the N-dimensional space of attribute values rather than the definition of a point set:

```

ATTRIBUTES { // list of attributes (defined similar to cells)
  attrName [ <- sourceAttr ] {
    //actual mapping from  $E^3$  to  $E^N$  (can reference external funcs)
    mapping ::= = MAPPING;
    // dimensionality of the attribute
    dim ::= OD..ND;
    :
    :
  }
}

```

The template IC is described next. The template IC can be used as a platform for the subsequent description of the instances (see section 3.8). Thus, an IC can reference a set of cells and attributes described previously (see figure 43). Additionally, descriptions of relations between the cells and attributes are provided, thus defining volumetric space partitions present in the model.

```

TEMPLATE_IC_NAME : TEMPLATE_IC {
  CELLS {...} // references to a set of previously described cells
  ATTRIBUTES {...} // the same for attributes
  // relations between cells and attributes
  ATTRIBUTES { ...
    // default attribute (for any point outside any of the defined
    // space partitions)
    default ::= ATTRIBUTE_NAME_X;
    // establish relations between the cells and the attributes
    RELATIONS { cellNameA attributeNameU; ... }
  }
}

```

The template IC can also include an enumeration of the topological and dependency relations specific to the model being described. This is a template of an IC that can be used as a platform for the subsequent description of the instances. Thus, this section includes references to a set of cells/attributes described previously and a description of relations between them^{9, 10} (see diagram in fig. 43).

```

RELATIONS { // relations between the cells from CELLS
  // list of containment relations (may provide names for them)
  containment { cellName1 cellName2 [relationName]; ... }
  boundary { ... // similar to containment }
  dependency { // list of dependency relations between parameters
    // dependency and its priority (optional - default being 1.0)
    // or a HIERARCHICAL specifier - may also provide names.
    cellNameA.paramNameX cellNameB.paramNameY [REAL | HIERARCHICAL]
                                          [relationName];

    // geometric dependencies
    cellNameC.shape cellNameD.shape [ REAL | HIERARCHICAL]
                                     [relationName];

    // attribute dependencies
    attrNameU.mapping attrNameV.mapping [ REAL | HIERARCHICAL]
                                         [relationName];

    // mixed dependency
    attrNameS.mapping cellNameT.shape [ REAL | HIERARCHICAL]
                                       [relationName];

    :
  }
} // RELATIONS

```

Two simplified examples of model definition demonstrating how a template IC can be used for an easier definition of a model are presented in sections 5.1 and 5.2.

⁹It is preferable to provide only a single description of the relations that remain unchanged over the course of the modelling process. Every instance can in its turn have its own set of relations valid over its lifetime.

¹⁰In certain models it could be useful to define a number of IC templates. The user could then introduce minor modifications to these descriptions in order to define new instances. This is similar to inheritance in object-oriented programming, with an additional option to remove components which are not present in any particular instance.

Finally, after all the preliminary steps the actual model description can be provided. As was mentioned earlier in section 3.9, this model description includes the definition of different states of the model reflected in IC instances. The definition of IC instances is similar to that of template IC (see fig. 43). The main difference being that the components of earlier introduced templates and IC instances can be used for the definition of new IC instances.

```

IC_INSTANCE_NAME {
  CELLS { // references to a set of previously described cells
    CELL_X CELL_Y; // reference earlier described cells
    USE TEMPLATE1.CELLS [ \ { CELL1 CELL2} ]; // from IC template
    :
    ATTRIBUTES {...} // similar to cells
    REACTIONS { ... } // custom REACTIONS of cells/attributes
    :
  }
  RELATIONS {
    dependency {
      // define new relations
      cellNameU.paramNameS cellNameV.paramNameT [ REAL ] [NAME];
      // use the IC template removing/adding relations
      USE TEMPLATE1.RELATIONS.dependency [\{relX ...}] [U relY...];
      USE instA.Rd[cellNameW, cellNameZ]; // use unnamed relations
      :
    }
    :
  }
}

```

The collection of parameters of cells/attributes that are considered important or meaningful within this IC instance is defined in order to provide a better understanding of the state of the model. Such parameters can also be defined for the whole dynamic model. These parameters can be used to track the state of the model and for high-level interaction with it¹¹.

```

STATE_PARAMETERS {
  stateParamsName ::= VECTOR(...) | REAL(...) ...;
  // new alias for nested parameters
  cellNameW.paramNameP newParamNameX;
  instanceParamNameU newParamNameY;
  :
}

```

As we mentioned earlier, custom reactions of IC entities can be defined with the description of IC instances. Finally, the predicate of an IC instance is defined in a way similar to reactions of the IC entities.

```

REACTIONS { // reactions to default events

```

¹¹This can be considered as one of the ways of parameterising the model.

```

init( REAL globalT) {...}
terminate( REAL globalT) {...}
update( REAL globalT, REAL localT, REAL lifeT, REAL dt) {...}
:
}
PREDICATE { // predicate of this instance
  bool evaluate( REAL globalT, REAL localT, REAL dt) {...}
}

```

Real examples of model definitions using the aforementioned syntax can be found in chapter 5.

It is worth noting that certain lower-dimensional cells and relations are introduced into the IC “implicitly”. This happens when lower-dimensional cells, boundary and containment relations can be evaluated automatically. For instance, a surface of a volumetric F-Cell and a boundary relation between its surface and an FRep object; edges and points of a mesh and the appropriate relations between them, etc. These automatically added cells can be referenced using a postfix. For instance, we may have a cell called `CellA` in 3D, its implicitly added cells (if any) can be referenced as `CellA2D`, `CellA1D`. Loading of the shape (which may be a mesh or an FRep model) leads to an implicit definition of the domain of the object (or a bounding volume around the object).

It is also worth mentioning that parameters have a special syntax for the assignment of their values. By default the assignment of a value to a parameter sets an absolute value. But it is possible to define a “relative value” using a suffix after the name of the parameter. For instance:

```
translate.relative = globalT * velocity12
```

This construction means that only the relative value of the `translate` will be modified (i.e., it will be added to the initial value of `translate` parameter). This can prove to be useful for the definition of motion/modification that does not itself depend on the initial state of the object. For instance, when the definition of the motion law does not require knowledge of the object’s initial position.

A similar approach could be used for shapes. When we want to find the

¹²Alternatively, a special assignment operator syntax could be used (e.g. “a #= b” or “a ^= b”).

global position of the geometry in space, we could use the “shape” property for the shape defined in local coordinates and “shape.global” for the transformed geometry in global space.

Additionally, there is an opportunity to define functions, which can be bound to IC instances and thus use components defined in them (e.g. to modify the parameters of the cells). Certain parts of the predicate of an IC instance can be identical to the predicate of another instance, thus an “attachable” function could be used to reduce definition duplication.

```
function partialPredicate()  
{  
  cell1.translate = ...;  
  cell2.shape = combine(cellX, cellY);  
}
```

In this example the function issued by the instance can gain access to the cells associated with it. If the referenced cells are not available, a compile-time error occurs.

This augmented notation should be sufficient to allow us to start creating descriptions of time-dependent hybrid models (see chapter 5 for a set of models described in this manner). These descriptions are parsed and converted into intermediate model descriptions which can be passed to the IC engine in order to evaluate the model over time. In the following sections we will provide a more detailed description of the algorithms and data structures used within the IC engine.

4.3.3 The process of model evaluation

Model evaluation is mostly hidden from the user. Users can customise certain steps using custom reactions, but the main process of model evaluation is the responsibility of the IC engine. The process of model evaluation can be fine-tuned and parameterised through the provided model definition. This is more of a declarative rather than an imperative description of the model. The main focus here is on the definition of all model components and on the relations between them. Reactions to events are also of a more declarative nature, as often they simply provide the descriptions of the dependencies between different properties of the IC entities over time. The generation and processing

events and the issuing of reactions and transitions between the instances is performed by the IC engine.

Below we outline a high-level view of the processes occurring inside the IC engine during the model evaluation process.

1. Update the global time of the IC using the parameters provided.
2. Call the update reaction of the IC (pre-update).
3. Update the local time of currently active instance.
4. Invalidate all the cells/attributes .
5. Evaluate the currently active instance.
 - (a) Issue a reaction of the IC instance to the modification of time (pre-update);
 - (b) For every cell/attribute of this instance (in the order defined by the dependency graph together with cell/attribute priorities):
 - i. Issue a reaction to the modification of time (pre-update),
 - ii. Evaluate the current transformations/deformations of the cell/attribute,
 - iii. Evaluate the new values of the time-dependent parameters contained in the cell/attribute,
 - iv. Evaluate the shape of the cell,
 - v. Issue a reaction to the modification of time (post-update),
 - vi. Issue a reaction to other events (if any occurred);
 - (c) Issue a reaction of the IC instance to the modification of time (post-update);
 - (d) Issue a reaction of the IC instance to other events (if any occurred);
 - (e) Evaluate the IC instance predicate/constraints:
 - i. If the IC instance is still valid, go to step **1**,
 - ii. If the IC instance is invalid, go to step **6**;
6. Call the custom update reaction of the IC (if it was provided).
7. Choose an appropriate instance from the dynamic IC:
 - (a) Evaluate the predicates of the available IC instances;
 - (b) Find a valid IC instance;
 - (c) Perform the transition from the previous IC instance to the new IC instance:

- i. Issue a reaction to the transition,
 - ii. Issue a reaction to the transition of all the active cells/attributes present in the new instance;
 - (d) Enable the new instance.
8. Go to step 1.

Some steps of this evaluation require additional detail as they present a number of issues, which need to be resolved. We consider these details in the next section.

4.4 The technical details of model evaluation

In the previous section we have outlined the high-level overview of the steps required for the evaluation of the model. In this section we discuss some of these steps in more detail expanding the description of the algorithms required for the correct evaluation of a dynamic IC-based model.

4.4.1 Dependencies and the order of evaluation

The dynamic IC framework provides the means of establishing different types of dependency relations of the dynamic entities within the model. These relations play an important part in a dynamic hybrid model, as they allow us to compose assemblies of objects and to create complex dynamic structures that change over time. We can define system components in a modular fashion and integrate these into various models of higher complexity through the application of dependency relations. Dependent entities are affected by their master entities, which in their turn may depend of other entities, thus forming complex dependency graphs reflecting semantic relationships existing within the conceptual model. State transitions of master entities automatically result in the modification of the state of their dependent counterparts. One of the challenging problems of model evaluation is related to the correct order of evaluation of the entities that affect each other. Special algorithms taking into consideration the mutual dependency relations between the entities need to be described. Otherwise we cannot guarantee the validity of the evaluation of

correctly defined dynamic hybrid models. Here we provide an overview of the issues relating to dependencies and discuss how these can be resolved.

The case of unidirectional dependencies

In a simple case only unidirectional dependencies are present in the model. Thus, we need to resolve a set of dependencies between dynamic objects. These are dependencies in which none of the dependent entities are used for the definition of the state of the master entity. According to equation 3 in section 3.5 this can be formulated as:

$$\forall i \in I_m^{i,X} \Rightarrow \forall j \in I_d^{i,X} : (C_j, C_i) \notin Rd_X$$

i.e. if the entity C_i is a master entity in any of the existing dependency relations, none of its dependent entities C_j is used as a master cell in any dependency relation where C_i is a dependent entity.

For instance, this could be defined in the following way:

```
dependency {
  cell_Ci.paramX cell_B.paramY; // cell_Ci is a master cell
  cell_B.paramY cell_Cj.paramZ; // cell_B is a master cell
}
```

It is worth noting that C_j does not affect the state of C_i even through a set of intermediate entities (i.e. through a dependency graph). In this case we can retrieve a set of acyclic graphs representing dependencies in the model. We then apply topological sort (Cormen *et al.*, 2001) to each individual dependency graph in order to define the evaluation order for each of the entities. The first node of every graph is then considered to be the root node. This root node of the graph is the main master entity of the dependency graph. For any entity in this graph we can find their “dependency depth”, i.e. the minimal number of entities that need to be evaluated before the evaluation of this entity can be invoked. We start the evaluation of the entities going from the root node of the graph down to its leaves. Entities with equal “dependency depths” can be arranged in a list of entities which can be evaluated concurrently. Additionally, each entity can be assigned a priority value to modify its default order of evaluation. This priority is taken into account when generating the list of entities which require an update. This list is initially generated depending

on the order of the entity definition and the provided dependency relations. Priorities are then used to locally sort cells within the “concurrent lists”¹³. Additional optimisations relating to a number of actual state transitions of the master entities affecting its dependent entities could also be considered. Cartwright discusses a number of ways which can be used in order to reduce a number of redundant evaluations keeping the model up-to-date considering all the dependencies (Cartwright, 1998). One of the proposed algorithms is the *block redefinition* algorithm, which allows us to minimise the number of model re-evaluations coupling a number of state transitions in a block instead of performing state transitions of the model whenever any state parameter is modified. A detailed overview of this topic is outside of the scope of this thesis.

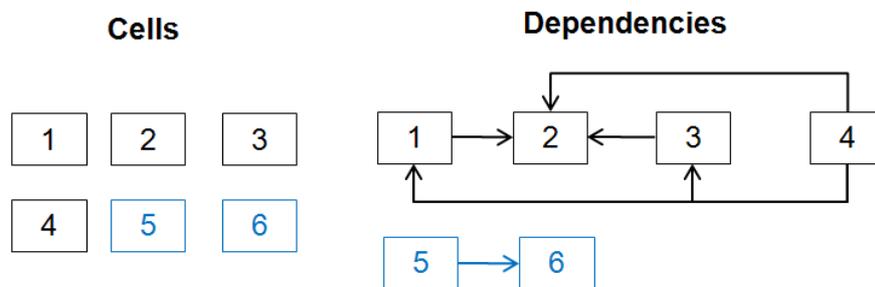


Figure 44: The initial set of cells and the dependencies between them.

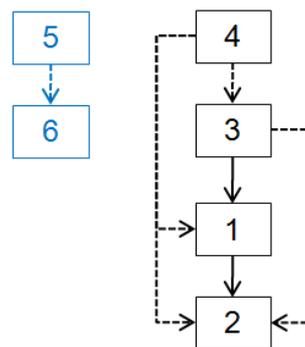


Figure 45: The lists of topologically sorted cells.

¹³Additional hints can be provided by the user in order to determine whether he/she wishes certain entities to be evaluated on the same processor or on a different one. This is similar to the usage of CPU affinity masks for every running process that can be set by the user on different operating systems. In theory concurrent cells could also be evaluated using a distributed architecture, which would be beneficial for IC models requiring a lengthy discretisation procedure.

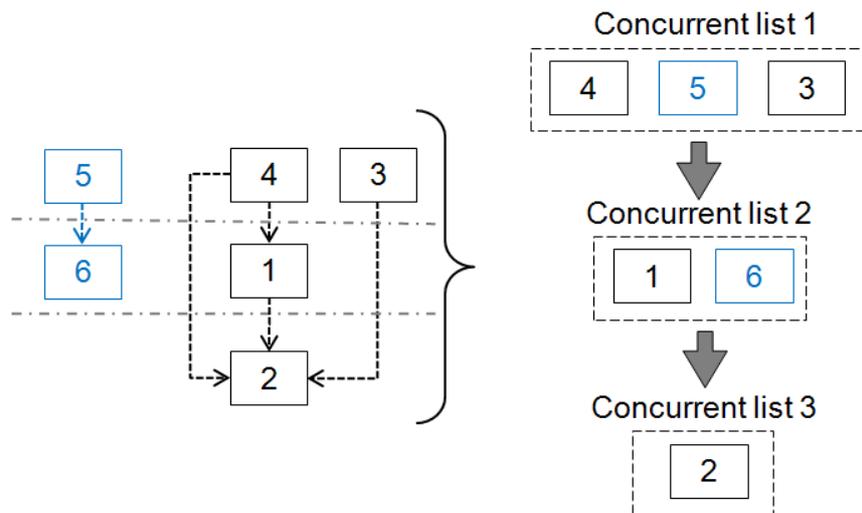


Figure 46: *The list of concurrently evaluated cells.*

A simple example illustrating an approach for the model re-evaluation process is presented in figure 44. Let us describe the dependencies of this model:

```

dependency {
  cell_1.param cell_2.param;
  cell_3.param cell_2.param;
  cell_4.param cell_1.param;
  cell_4.param cell_2.param;
  cell_4.param cell_3.param;
  cell_5.param cell_6.param;
}

```

In this example we have a set of six cells and a number of dependency relations between them. For instance, `cell_1` is a master cell for a dependent cell `cell_2` and `cell_4` in its turn is a master cell for a dependent cell `cell_1`. From the illustration we can see that there is a complex combination of dependencies between the cells **1** to **4**. In order to resolve the order in which these cells need to be evaluated we apply the topological sort algorithm mentioned earlier (see fig. 45). The resulting sorted dependency graphs are used to generate lists of entities that can be evaluated concurrently (see fig. 46).

The case of bidirectional and circular dependencies

The problem of dependencies becomes even more complex, if bi-directional or circular dependencies are allowed. In this case the topological sort mentioned earlier cannot help us resolve the order of evaluation. According to

equation 3 in section 3.5, the presence of circular dependencies can be formulated as:

$$\exists i \in I_m^{i,X}, \exists j \in I_d^{i,X} : (C_j, C_i) \in Rd_X$$

i.e. if the entity C_i is a master entity in any of the existing dependency relations, any of its dependent entities C_j may be used as a master cell in a dependency relation where C_i is a dependent entity.

In the simplest case the presence of circular dependencies in a model could be forbidden in order to avoid ambiguities. But bi-directional or circular dependencies can also be a powerful tool for the definition of dynamic IC-based models. Before going into a description of the issues arising from the presence of circular dependencies, let us consider a simple example illustrating the usefulness of this type of dependencies.

Bi-directional dependencies can prove their usefulness in a model with two or more interacting objects ¹⁴. Figure 47 demonstrates the simplest case of Newton's cradle. This cradle consists of two pendulums. When the two balls at the ends of these pendulums collide, they exchange their momenta. This system could be defined within the dynamic IC framework using four IC states (fig. 48). The first IC instance reflects the state of the model when there is no interaction between the objects and only the first ball on a thread is moving. The second IC instance reflects the moment when the two balls touch each other and the impulse of the first ball is transferred to the second one. In the third IC instance only the second ball is moving, while the first one remains static. The fourth IC instance is where the two balls are touching again. This time the impulse of the second ball is transferred to the first ball. Impulse transfer can be modelled through a unidirectional dependency relation between the moving and a static ball. So that the velocity of one of the balls is set to be equal to the velocity of the ball that is represented as a master entity within the dependency relation.

The definition of four IC instances for such a simple model is unnecessarily complex and excessive. The description of the model would become even more redundant with the increase of the number of pendulums present

¹⁴Here we refer to what is called a coupled systems in physics.



Figure 47: A simple model of Newton's cradle.

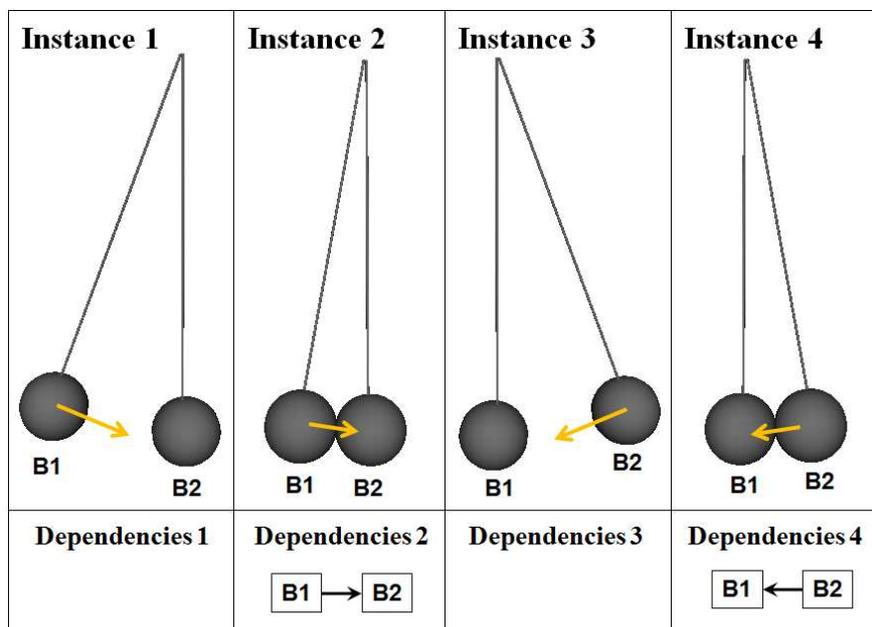


Figure 48: The four states of Newton's cradle.

in it. This is due to the fact that each state of the model has to reflect the distinct behaviour of each individual component within it. The description of the model we have presented above does not adequately reflect the nature of the model, but it tries to mimic the behaviour of the entities at different periods of time. Alternatively, we may apply the concept of bi-directional dependency in order to provide an alternative definition of the model. This new description only has two IC instances defined in it (fig. 49).

The update of the first IC instance leads to the change of the position of both balls using their current velocities. In the second IC instance, a transfer

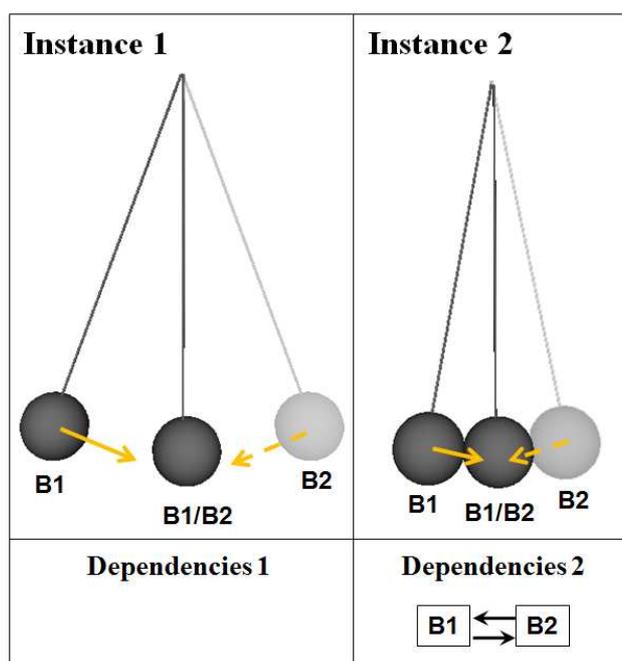


Figure 49: *The two states of the cradle: (Left) One of the balls moving, no collision (Right), One of the balls moving (collision situation).*

of momenta of the two balls needs to be performed. The transfer of momentum simply means that the velocities of the balls need to be exchanged at the moment of time when they collide (and optionally reduced due to natural energy losses). In terms of the dynamic IC framework, this means that there is a dependency relation between these two entities, i.e. parameter value of each entity depends on the parameter of the other entity. Thus, the set of dependency relations (R_{dp}) for this model consists of two pairs:

$$R_{dp} = (B1, B2) \cup (B2, B1)$$

As can be seen this IC model has a circular dependency between cells **B1** and **B2**, as both **B2** depends on **B1**, while **B1** is being a master cell for **B2** at the same time. The presence of circular dependencies poses the following questions which need to be answered:

1. Which of the cells participating in a bi-directional dependency relation should be evaluated first?
2. Which value should be provided to the dependent cell if the value eval-

uated by the master cell depends on some value of the dependent cell?

In fact, the same questions arise when we have a more complex model where there are circular dependencies created through a chain of dynamic entities. Some of these issues could be resolved automatically. For instance, the cells participating in circular dependencies could retrieve values reflecting the previous state of the cells they depend on, though the user should be given the option to define the desired behaviour of the system in such situations. One of the ways to achieve this is to use the aforementioned priorities. This means that each dependency relation should be defined along with its priority. If the priority is not defined explicitly, a default value is assigned to each dependency relation.

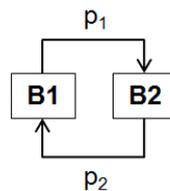


Figure 50: A simple bi-directional dependency.

Let us consider the pendulum example shown, in figure 49, in more detail taking its dependency relations into consideration. Figure 50 depicts the dependency relation between the cells **B1** and **B2**:

```

dependency {
  B1.paramX B2.paramY p1;
  B2.paramV B1.paramW p2;
}
  
```

In this example cell **B2** depends on cell **B1**. The priority of this dependency relation is p_1 . Cell **B1** in its turn depends on the cell **B2**. The priority of this relation is p_2 . This circular dependency can be resolved if we distinguish between the different states of the cells at different moments of time. Fig. 51 illustrates two different evaluation orders depending on the relation priorities. If both priority values are equal, a higher priority value is assigned to the dependency relation which was defined first.

Another option is to allow the user to request the re-evaluation of the cells using all up-to-date states (fig. 52). In this case states of the cells need to be evaluated more than once in order to reflect all changes occurring in the

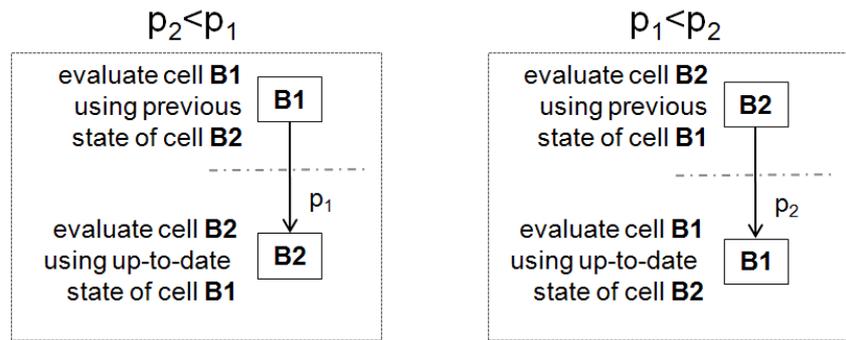


Figure 51: *Two different evaluation orders.*

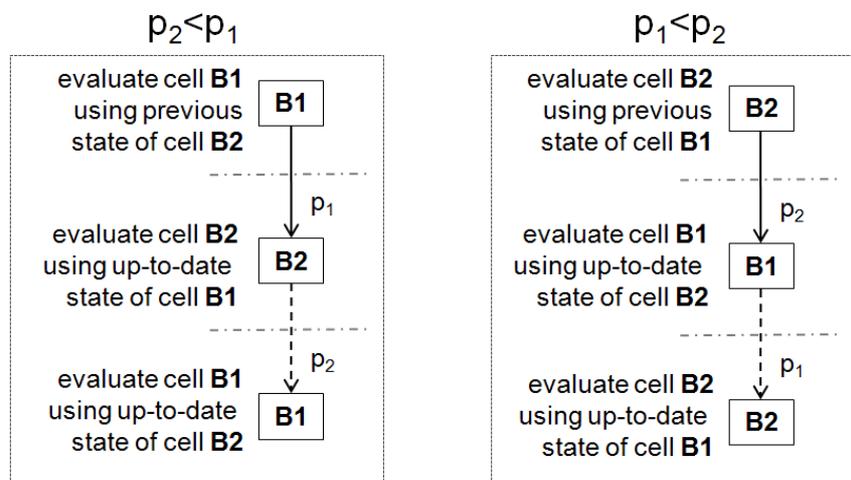


Figure 52: *Two different user-controlled evaluation orders.*

model as a reaction to an event. In the “Newton’s cradle” example, it is sufficient to use previous states of the cells only. In the second IC instance, ball represented by the cell **B1** retrieves the velocity of ball **B2** (implicitly the velocity from previous state is provided), while ball **B2** retrieves the velocity of ball **B1** (also using the previous state). Positions of both balls are updated accordingly (see fig. 49). After the evaluation of the predicate of this IC instance, a transition is performed to the first instance, as there are no further interactions between the two balls. Only this time the initial velocities of the balls are different, while description of their behaviour remains unchanged.

As we have mentioned before a topological sort cannot be applied. This algorithm can be modified in order to resolve the dependencies in a model with existing cyclic dependencies. We perform a modified topological sort algorithm using priority values to “cut” the graph in order to retrieve the acyclic

dependency graph giving preference to the relations with the higher priorities. The cells involved in a cyclic dependency relation with a lower priority are evaluated using the previous state of the cells they depend on or by issuing a second evaluation with up-to-date states as outlined above.

Let us illustrate this approach with a more complex example of cyclic dependencies in order to illustrate how the evaluation order of the entities can be determined. We modify the example shown in figure 44 by adding an additional dependency (see. fig 53). Now this dependency graph has a cycle in it. In order to build an acyclic graph from it we use the values of the dependency relation priorities. Figure 54 depicts two possible evaluation orders for this case. The dashed cells refer to previous states of master entities.

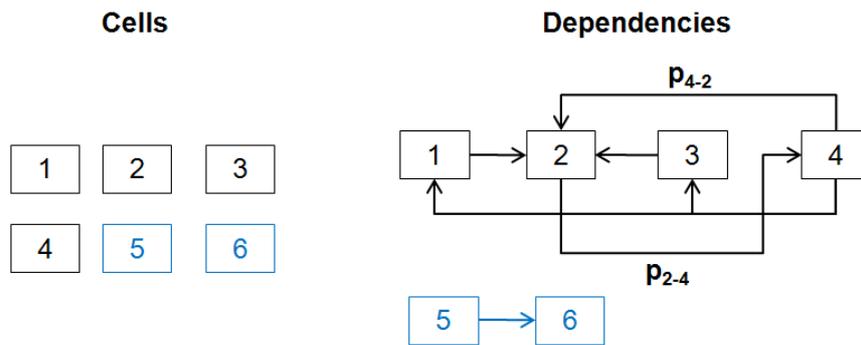


Figure 53: The set of cells and dependencies between them.

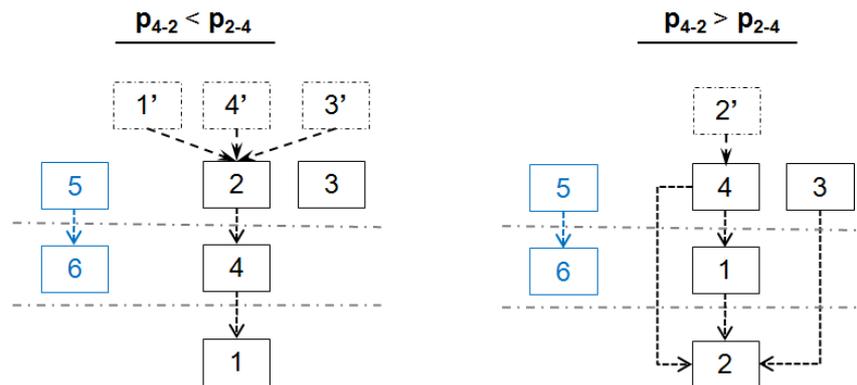


Figure 54: Two different user-controlled evaluation orders.

More detailed information and possible solutions to the aforementioned problem are outlined in (Cartwright, 1998; Adzhiev and Beynon, 1999). Cartwright (Cartwright, 1998) additionally accounts for the improving performance of

model evaluation through the application of a *block redefinition* algorithm, which allows him to minimise the number of model re-evaluations coupling a number of state transitions in a block, instead of performing the state transition of the model whenever any state parameter is modified.

A more elaborate example involving circular dependencies between mixed-dimensional components of a time-dependent model will be provided in the next chapter.

Additional tools related to dependencies

The dependency graph built after the establishment of the dependency relations between the properties of the objects can be visualised to help the user gain a better understanding of the dynamic model structure (fig. 55).

When dependency relations are established between properties, the values of the properties of the master entities are automatically reflected in their dependent properties. Thus, a dependent entity does not need to be aware of which precise cell it depends on. Dependent cells only request a value for a property which can either come from a master entity, be defined by the user during the modelling process or be a default value set at the initialisation phase. In the end, the behaviour of an entity is defined only using its properties and we do not need to modify it every time a dependency is changed. This allows us to “localise” the behaviour of an entity, so that it can be put into a different model without being fully aware of the context of its surrounding. On the other hand if such knowledge is required, an implicitly defined dependency can be provided within the reaction of an entity. This is achieved through the definition of a behaviour of a cell or of an attribute using parameters of other entities “visible” to the entity¹⁵.

Another concept which may prove to be useful is that of visual debugging. A selected set of properties of the IC entities could be visualised automatically. These values can be shown next to the cells or attributes (as numbers or rendered as vectors) or printed using dynamic spreadsheet. This technique could be useful for reflecting the changes of the property values over time. For instance, the velocity or the acceleration direction could be rendered next

¹⁵“Visible” refers to a set of entities being active within current IC instance.

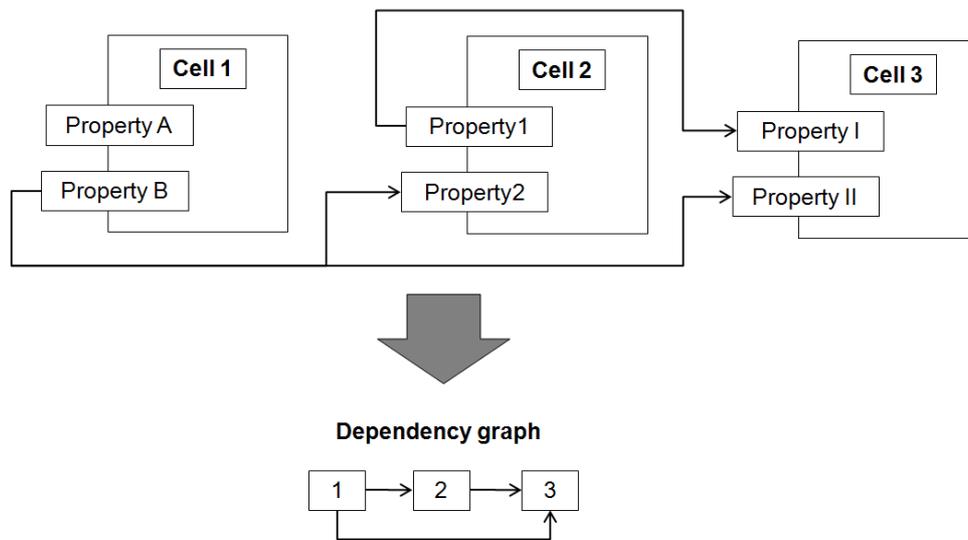


Figure 55: *The dependencies between the properties and the dependency graph between the cells.*

to a cell as well as the path travelled by the cell or its trajectory defined by the animation curve. These features require additional consideration and need to be further investigated in the future.

4.4.2 Handling an event requesting a modification of time

In section 4.3 we mention that custom events can be defined in an IC model. In the majority of cases these events are used within the model as a result of meaningful state transitions. These events are only processed by the entities defined within the model. The IC framework automatically fires time modification events in order to signal that time has changed and all components of the model need to update their state. As we have mentioned in the description of the framework, a reaction to the modification of time is invoked before all states of the entities were validated (“pre-update”) and after all the values have been validated (“post-update”). The required time step can be defined by the user during the modelling session. It is also important to provide a way which allows for specific modifications of time controlled by the entities present in the model. This is an important feature required for simulation applications (Witkin and Baraff, 1997), where the required time step needs to meet the simulation requirements. Otherwise significant precision errors will

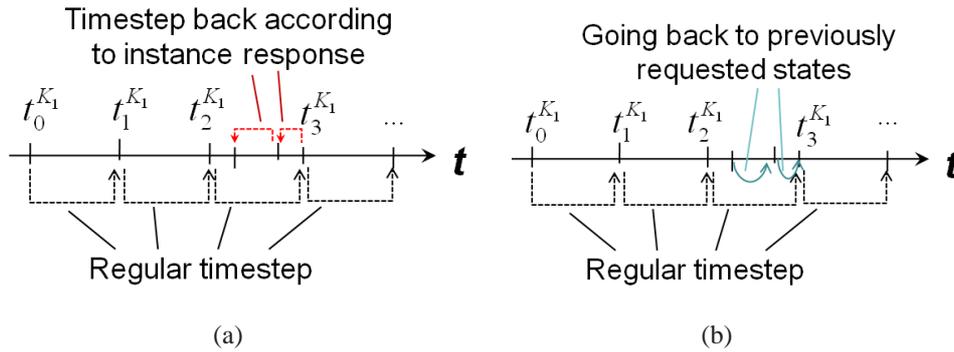


Figure 56: *On-demand adjustment of time (a) Time-step back in time (b) Going through all the previous states.*

be introduced to the model which may lead to incorrect simulation results. In the next chapter we also demonstrate an application where a time-step adjustment of the model allows us to improve the results of space-time modelling (see section 5.3.4).

Thus, we provide a mechanism allowing any entity to notify the IC framework of a required modification of time. Entities can notify the system whether they “accept” or “reject” the time currently set in the model. In response to this, the IC framework retrieves the required time-step and modifies the current time unless a newly set time is accepted by all active entities. The notification of the IC framework is achieved through the generation of an event by the entity (see fig. 56a). The framework then analyses the event. If the change of time was requested and an exact time-step was defined, it waits for the end of all the evaluations being performed at the moment (i.e., concurrent update of other cells), it saves the current state of the IC instance (i.e., all the entities whether they have already been validated or not) and it updates the time, restarting the whole update procedure again. After the evaluation at the requested moment of time, a valid instance needs to be evaluated at the moment of time originally requested by the system (fig. 56b). That is the reason why the state of the IC instance needs to be saved beforehand. This is needed in order to satisfy the requirements of the external system that could have requested the state of the IC model at this particular moment of time. This could be the rendering system reflecting the state of the IC using a specific frame rate.

A textual description of the request for the modification of time is provided in a very simple way:

```
// fire special event
// provide a relative step from current moment of time
IC::fireEvent(TIME_REJECTED, newTime);
```

We use a function provided by the IC framework, which accepts the type of event required to occur and a set of parameters of this event. In this case the parameter is a relative time step from the current time instant. This mechanism is used for the definition of non-linear sampling in the space-time domain described in section 5.3.4.

If during the evaluation process there was a request to step back to a moment of time earlier than the last correctly evaluated time, the evaluation procedure is terminated. Such a dynamic IC model is considered invalid because of its non-deterministic behaviour. Let us demonstrate this by using the example shown in fig. 56a. Here the model was successfully evaluated at the moment of time $t_2^{K_1}$. The IC framework increased the time to the moment of time $t_3^{K_1}$. One of the entities requested a step back in time a number of times. If the newly requested time is before the moment of time $t_2^{K_1}$, the provided IC model is considered to be invalid.

4.5 A brief description of the IC API

In previous chapters we have presented a description of the dynamic IC framework. We have also described a high-level notation which could be used for the definition of the actual hybrid models. In order to evaluate the IC model using its textual definition we need to perform a translation from this definition to a set of programming terms. This means that we need a programming interface allowing us to map the conceptual model to a computer model, which can then be evaluated by the IC engine using the algorithms described in this chapter. This programming interface is called an IC API (Application Programming Interface). As mentioned earlier in the text, ICs allow the incorporation of an object defined in a number of diverse representations. The behaviour of such objects can be defined using predefined animations, procedural descriptions or a mixture of both. This means that the IC API has to

support a number of techniques for static and dynamic modelling. Figure 57 illustrates the structure of the underlying components of the IC API allowing us to achieve this.

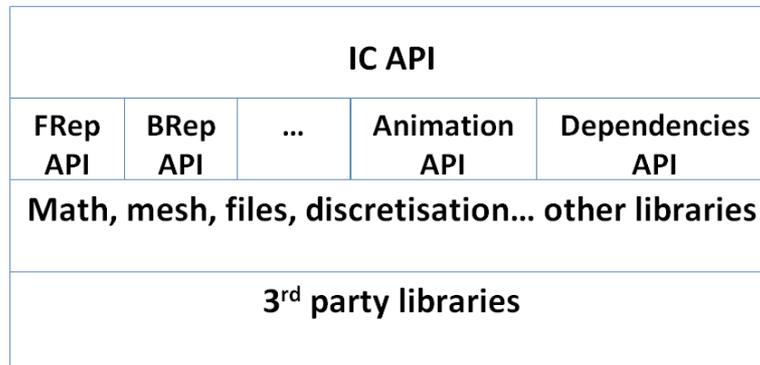


Figure 57: *The structure of the IC API and related tools.*

The IC framework allows us to incorporate a number of the existing representations within one model. Thus, it requires the support of FRep, BRep, PRep and possibly other representations. Primitives and operations specific to each representation are implemented as separate independent components. The animation API allows us to provide additional functionality for the definition of complex animation sequences of heterogeneous objects. Additional component implementing IC relations and dynamic dependency relations helps us to build compound relations between the objects in a dynamic hybrid model. The IC API uses all these components in order to accommodate hybrid models in a unified way and to provide the user with a set of tools required for the actual modelling. IC API allows the user to mix models of different representations using the tools available for each of them.

In section 4.2 we have outlined the main entities incorporated in the IC. The IC API reflects all the terms introduced in this thesis and provides a way of working with IC entities in order to be able to define and evaluate the model. The object-oriented paradigm was chosen as one of the appropriate approaches for implementing this framework. Each type of entity is mapped onto a separate class in the API. These classes and relations between them are depicted in figure 58 (see fig. 38 in section 4.2 for an overview of the components of IC entities). Each class has a set of properties and methods to manipulate the entities.

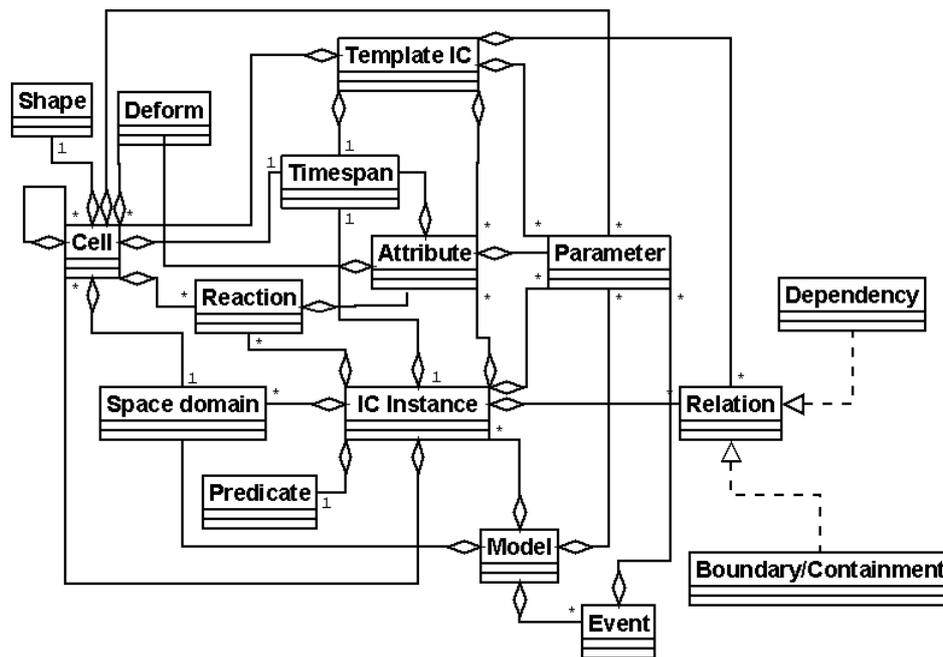


Figure 58: The UML diagram of the main classes present in the IC API.

The high-level UML class diagram (Booch, 2004), shown in figure 58 reflects the relations between the main classes of the IC API. As was outlined earlier a `Cell` encapsulates a `Shape`, a `Deformation`, a `Timespan/Life span`, a `Space domain`, sets of `Parameters` and `Reactions`. An `IC Instance` includes sets of `Cells`, `Reactions`, `Attributes`, `Parameters` and `Relations` as well as a `Space domain`, a `Predicate` and a `Timespan/ Life span`. Finally, a `Model` is composed of `Instances`, and `Parameters` reflecting the state of the `Model` and `Events`, occurring in the model at any moment of time, and a global `Space domain` within which the model is defined.

A more detailed description of every entity and its components is presented in table 2 in Appendix A. All the components present in this table are available to the user and can be set-up accordingly. All the entities reflected in the API are provided to the user in order to allow him to directly map a theoretical description of the model, defined in chapter 3, to an actual model definition that can then be evaluated. But the main idea of providing the API is the abstraction of the exact application that the IC API is used for. This means that the functionality provided by the IC framework can be integrated

into different applications.

The model can be defined explicitly using the API implemented for a general purpose object oriented programming language. We have chosen C++ (Stroustrup, 2000) as the main programming language used for the implementation of the IC API. C++ is still one the most widely-used programming languages, providing a good trade-off between the high-level features present and the performance that can be achieved with it. Another important factor is the availability of a large number of third-party libraries allowing the easier integration of existing static and dynamic modelling techniques (Overmars, 1996; Cignoni *et al.*, 2008; McNeel and Associates, 2010; Lavoie, 2010; Junker, 2006).

A domain specific language (see section 4.3.1) can be used to provide a higher level description of the model, while a translator performs the mapping from the DSL to the C++ IC API. This DSL can be implemented using ANTLR (Parr, 2007) for both the lexer and the parser. The output can then be a set of appropriately set-up IC data structures or an abstract syntax tree (AST) describing the IC-based model. An IC-based model can then be built using this description. The evaluation of the AST could be performed using a simple stack-based virtual machine with additional memory storage. Instead of our own implementation of this VM, the Low Level Virtual Machine (LLVM) infrastructure could be used (LLVM Developer Group, 2010). The LLVM can help us significantly reduce the development times and to improve the performance through its run-time compilation to native code (see section 4.6.8).

A special modelling environment is needed in order to provide a powerful way of working with the model. This environment should include a way of defining and refining a model on the fly, which assumes the availability of a discretisation and rendering engines reflecting the states of the model over time, along with an advanced scripting engine allowing the user to modify the model description (see figure 59). The process of model definition and analysis can be simplified using a set of visual metaphors reflected in a specialised Graphical User Interface (GUI). The complete modelling environment based on the IC API is depicted in figure 60.

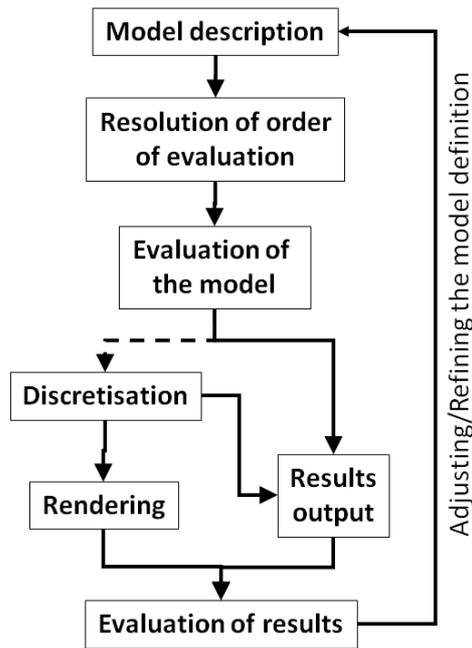


Figure 59: The IC modelling work flow.

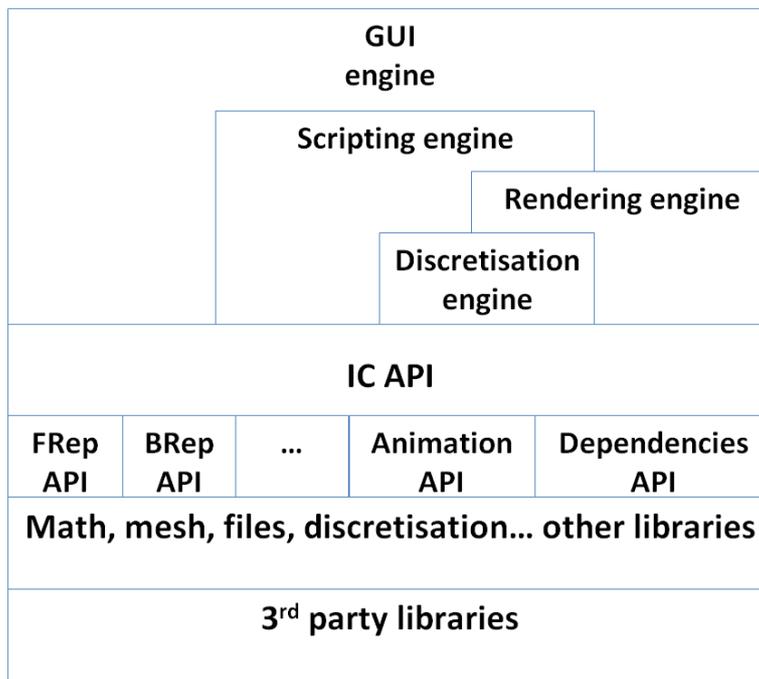


Figure 60: Full structure of the modelling environment and basic tools for IC modelling.

4.6 The FRep API as a subset of the IC API

In the previous section we have described the IC API allowing us to perform modelling using the dynamic IC framework. The dynamic IC framework in its turn incorporates a number of the existing representations. The framework has to provide full support for the majority of the existing representations in order to allow the user enough flexibility for the definition of diverse hybrid models. A full implementation of the framework requires significant human and time resources. Thus, a full implementation of the framework is outside of the scope of this thesis. Here we specifically focus on the implementation details relating to the definition of the underlying FRep models incorporated in a hybrid model. We have already mentioned that the FRep is a very powerful representation in its own right, which is highly suitable for the definition of dynamic heterogeneous objects (see section 2.2.3). Hence, support of FReps in the IC framework significantly enriches the feature set available for the definition of dynamic multi-dimensional heterogeneous objects. Another important reason to pay particular attention to the implementation of FRep components, within the IC framework is the absence of a common FRep toolkit, which would allow us to take advantage of all the significant aspects of this representation. Unlike the case of BReps and certain types of PReps, there is only a limited set of tools available for FRep modelling. These tools have been developed for a number of specific applications and cannot be used outside of them. These FRep-related tools were not designed for integration into other applications and cannot easily be plugged into an IC framework. This effectively means that there is no common way to work with FRep models outside of the set of existing tools (more details are provided in section 4.6.1). In order to be able to define FRep models we need to provide an FRep API allowing us to define Hypervolume objects which can be integrated into our hybrid model.

In this section we will describe the existing approaches to FRep model definition. We then propose and describe our novel FRep API allowing us to work with FRep models within the IC framework or independently. We present details regarding our design decisions, implementation details, case studies and specifics of the integration of our FRep API into the IC API. Fi-

nally, we describe an easy way of multilevel extensions introduced to the FRep API together with possible performance improvements of the FRep model evaluation.

4.6.1 HyperFun

At the moment the main modelling tool used for the creation of FRep models is a high-level programming language called **HyperFun** (Adzhiev *et al.*, 1999). **HyperFun** is a C-like language supporting a set of built-in FRep primitives and operations. The available set of FRep entities can be extended by the introduction of new primitives and operations to the core FRep library. Another way to extend the modelling system is to define new entities using the **HyperFun** language. HyperFun is an interpreted language. It is thus a platform independent language (obviously one needs to compile the interpreter itself for the target platform). The interpreter can also be integrated into external applications if required. **HyperFun** is a powerful language allowing us to describe complex Hypervolume objects in multidimensional space. Unfortunately it has a number of drawbacks as well:

1. The interpreter is strongly tied to the part of the program that constructs the model tree. It is hard to create and manipulate an FRep model without actually describing it in the **HyperFun** language. There is no intermediate layer between the internal model representation and the interpreter building this representation from the textual description of the model. Such an intermediate layer could allow third party applications to create or modify the model in a unified way. This problem also complicates the process of FRep modelling.
2. As **HyperFun** is an interpreted language, it is relatively slow (Uhlir and Skala, 2003), which is especially noticeable with complex models. Even though the textual model definition is parsed only once and converted into custom byte code, evaluation of the FRep models requires this byte code to be executed a large number of times. Direct compilation of an FRep model to platform specific native code could significantly decrease the time needed for model evaluation. It is also important to note that **HyperFun** interprets each definition of the model

irrespective of the results of previous evaluations. In theory tracking the changes between different iterations of the working process with the same model could be used to determine which parts of the model require re-evaluation. Partial model evaluation is also crucial for interactive modelling, so that the model is subsequently evaluated only in the areas locally modified by the user.

3. **HyperFun** was designed to be a simple lightweight language. Users are mostly working with functions, using a set of simple built-in geometric primitives and operations. Each of these functions is evaluated for each point in the modelling space. Thus it is not possible to define an object which would require a particular type of pre-processing that only needs to be performed once. The definition of complex models requiring non-trivial space transformations and inter-object relations forces the user to think of the model in terms of a constructive tree. An object-oriented approach on the other hand could hide some of these concepts and let the user work with the model at a higher level of abstraction. FRep entities could be manipulated through the modification of their properties and combined together creating new entities. At the same time, there could also be another intermediate layer between the model and the user allowing him/her to manipulate the model at a lower level.
4. The **HyperFun** language can be used for multidimensional modelling. The user can get access to an arbitrary number of “multimedia” coordinates, but FRep entities within the model do not have an associated property characterising their dimensionality. Moreover, the library of FRep primitives is limited to only 3-dimensional objects. This over-complicates the process of mixed dimensional modelling.

From the above enumeration of the most notable limitations of **HyperFun** it is clear that we need a more general and extensible way for the FRep model definition. We need to directly map all the existing formal FRep concepts to a programming paradigm, so that every part of the FRep framework could be available to the end-user, regardless of the application area and the type of the problem being addressed. We call such a programming framework the FRep

Application Programming Interface (FRep API).

4.6.2 Mapping FRep concepts to the FRep API

The basic structures in this section are defined in UML (Booch, 2004) and can be implemented in various object-oriented programming languages. However, in this work we decided to use the C++ programming language for our implementation, as was explained in section 4.5. C++ can also be more easily tied to other applications through the use of dynamic linking. This is important both for the integration of the API and its on-the-fly extension. In that way even FRep entities defined in a different programming languages can be used together.

The mapping of all the FRep concepts to programming terms allows us to flexibly define FRep models in a unified way. Concrete FRep models can be built directly from a theoretical description. Such a capability can be used by a new interpreted language supporting the entire set of FRep features (both existing and those that may be introduced in the future), a custom interactive FRep modeller or a third party modelling software packages and a wide range of other applications that could benefit from the use of FRep models. Every FRep entity (be it a primitive, an operation or a complex object) needs to be directly mapped to the FRep API. We intend to allow the user to manipulate any entity in a unified way and also to have access to its custom properties.

The FRep entity

All concepts available in the FRep and Hypervolume modelling theoretical frameworks are mapped to a set of technical terms, in order to provide a transparent way of model definition using the API. We start the FRep API description with the introduction of a base interface used by all the actual FRep entities present in the model. The basic model definition and manipulation is done using FRep entities. These entities are inherently functions with a set of additional properties and methods.

An `ENTITY_BASE_T` is an interface reflecting all the properties common to FRep entities (see fig. 61). Since all of the entities can be represented as functions, each entity has an `evaluate` method which retrieves a coordinate

vector (input parameter) defined in the modelling space of the appropriate dimensionality, performs a custom function mapping (evaluation), based on its internal parameters and provided coordinates, and returns its result for a specified dimensionality (fig. 61). This method can have more than one instance allowing us to evaluate all possible intervals of function values within certain area using interval or affine methods (Junior *et al.*, 1999; Flórez *et al.*, 2008; Knoll *et al.*, 2009; Fryazinov *et al.*, 2010). This may be needed in order to estimate the function values within provided region of space.

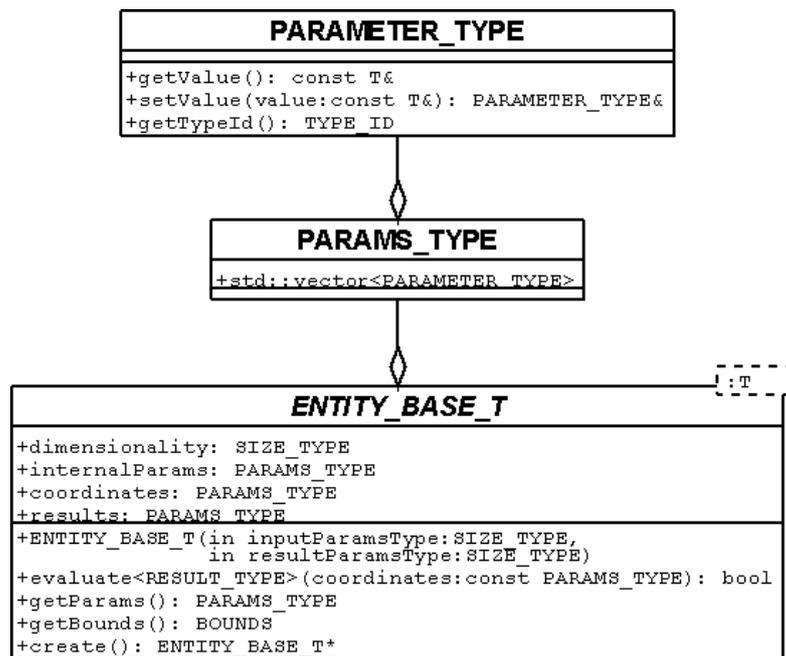


Figure 61: A simplified UML diagram of an FRep entity

All the important properties of an entity are stored in its parameters. The **PARAMETER_TYPE** is an important data type which allows us to manipulate values of different types in a uniform way. For instance, the radii of an ellipsoid or the line segments of a convolution surface can be stored and manipulated in the same way. The user-defined data types can also be wrapped around. The **PARAMETER_TYPE** also allows us to distinguish between different data types stored within it, which makes it possible to perform robust runtime type checks validating the data structures (i.e. guaranteed type-safety) and the connections established between them. Another important factor regarding the exposed universal parameters is that they can be used within a hybrid model. Parameters of entities defined in an FRep model can be manip-

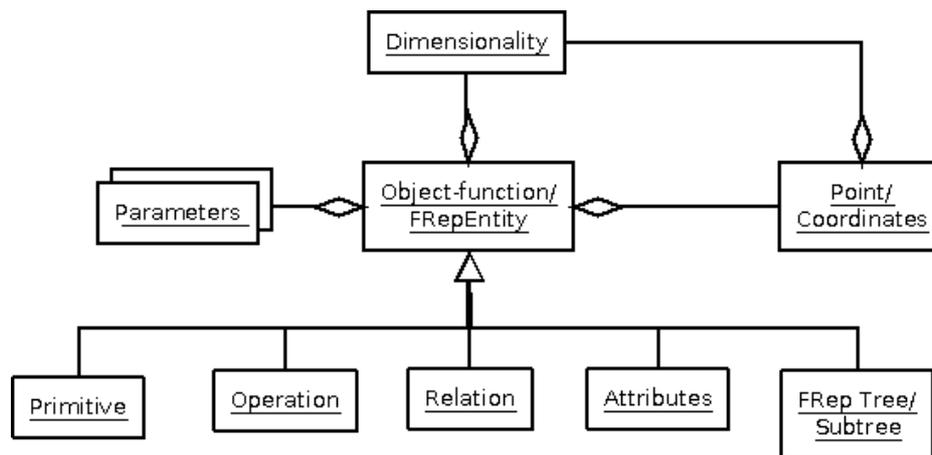


Figure 62: *The high-level overview of the types of FRep entities.*

ulated and shared by entities specified in different representations in a unified way (see details of integration of the FRep API into our IC API in section 4.6.7). For instance, the line segments defining convolution surfaces can be driven by the skeleton used for the animation of a polygonal characters. Another example could involve a non-linear fitting procedure, which modifies the weights of distinct convolution surfaces in order to embed the resulting FRep geometric object into a polygonal shape (see. details in 5.4). There is also the possibility of coupling traditional computer animation techniques to define the values of particular parameters over time. These values can subsequently be interpolated using one of the existing interpolation techniques. Alternatively values of these parameters can be defined in a procedural manner through the same unified interface. This complies with the requirements for IC entities defined in a hybrid model.

Figure 61 only reflects the main characteristics of the ENTITY_BASE_T class. Additionally, each entity provides a method for analytical intersection with a ray (if such method is available for this type of entity), a method for classification of the entities (see figures 62, 63 and 64) and some other functionality required for flexible manipulation of FRep models.

As explained in section 2.2.3, there are a number of fundamentally different entity types available in FReps. We will provide a description of each of these outlining their specifics.

The FRep primitives

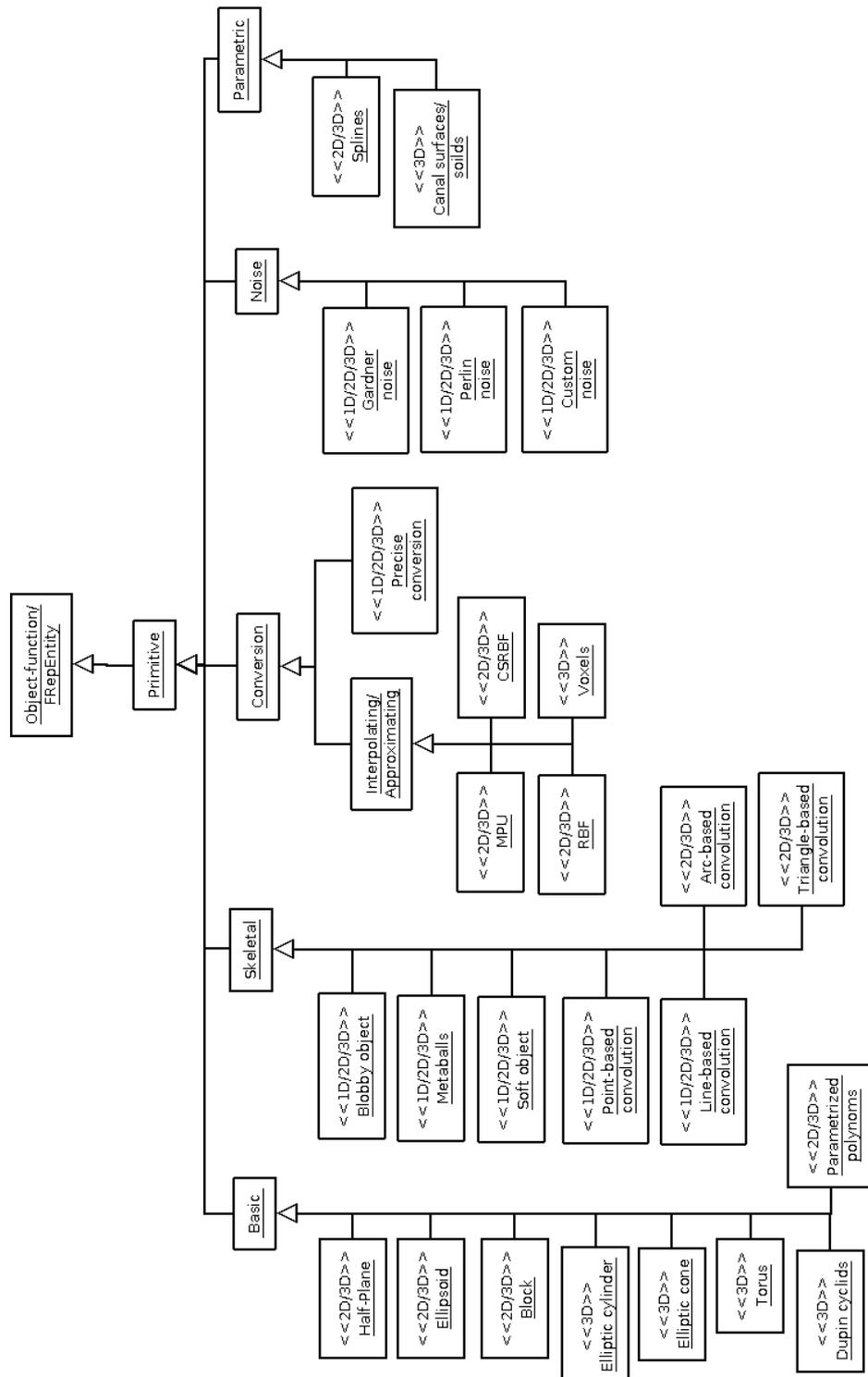


Figure 63: Types of primitives available in the FRep API.

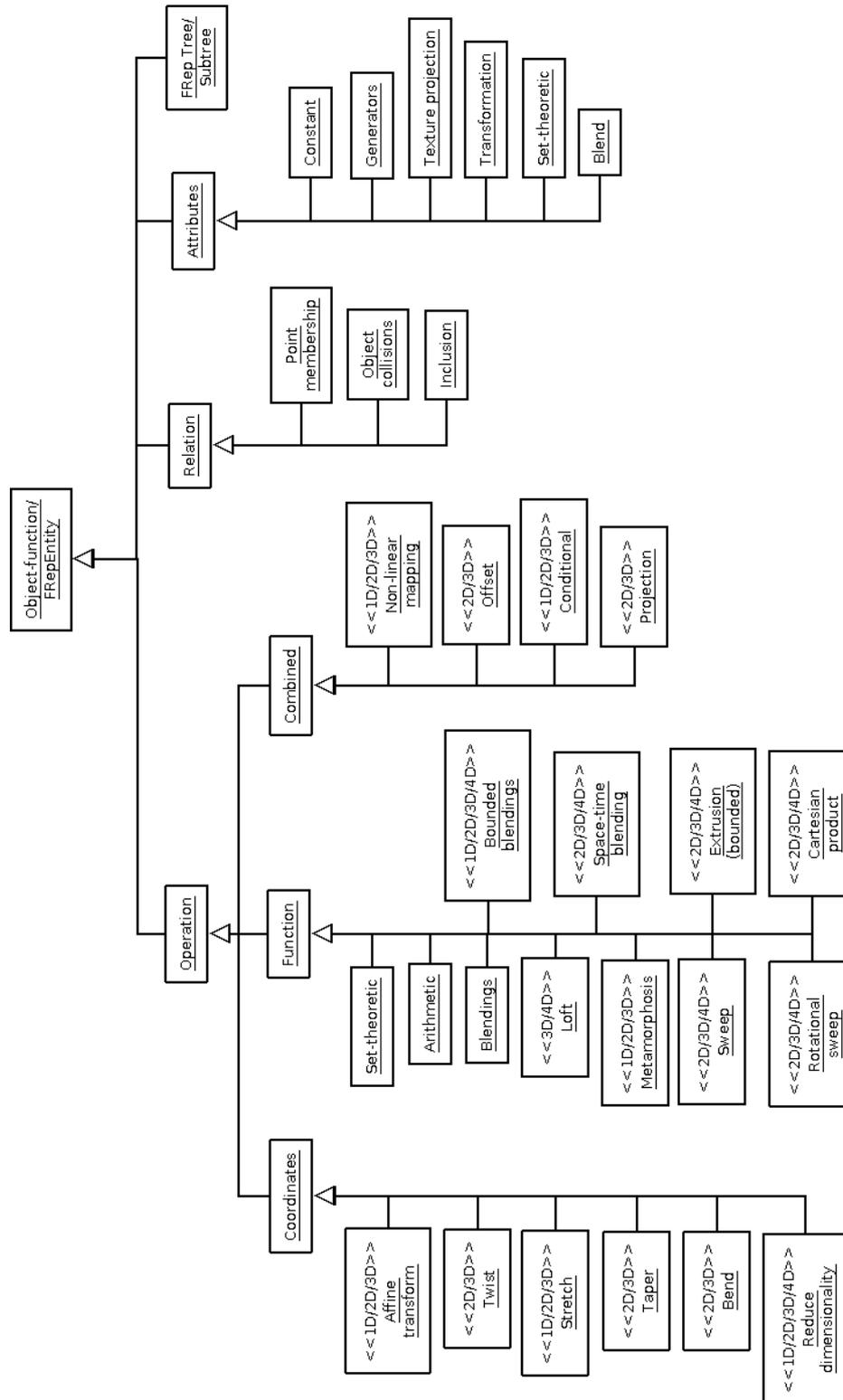


Figure 64: Types of entities available in the FRep API.

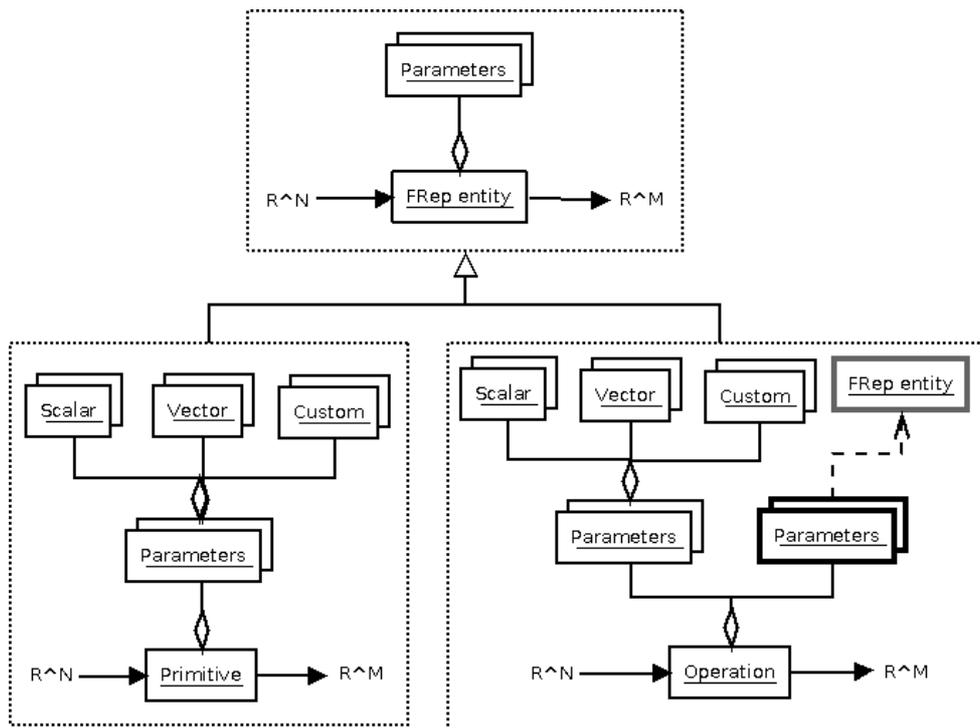


Figure 65: *The difference between a primitive and an operation.*

An FRep primitive simply returns the result of the functional mapping associated with it. This result only depends on the values of the internal parameters of a particular primitive. These can be simple parameters of algebraic surfaces (e.g. the radii of an ellipsoid or the dimensions of a box) or sets of parameters calculated by the primitive during a pre-processing step (e.g. the coefficients used by interpolating implicit surfaces). The list of available primitives is shown in fig. 63.

The FRep operations

Unlike FRep primitives FRep operations required a set of dependent input parameters, i.e. values of these parameters depend on the result of the evaluation of other FRep entities, be it primitives or operations (see fig. 65). An FRep operation initiates a request for the evaluation of the entities it is applied to (see fig. 66). An FRep operation does not depend on the type of FRep entity it works with, as it only needs to retrieve the values resulting from the evaluation of the entity it is applied to. The coordinates provided to other evaluated entities can be modified. That is how affine transformations or non-linear space mappings are implemented.

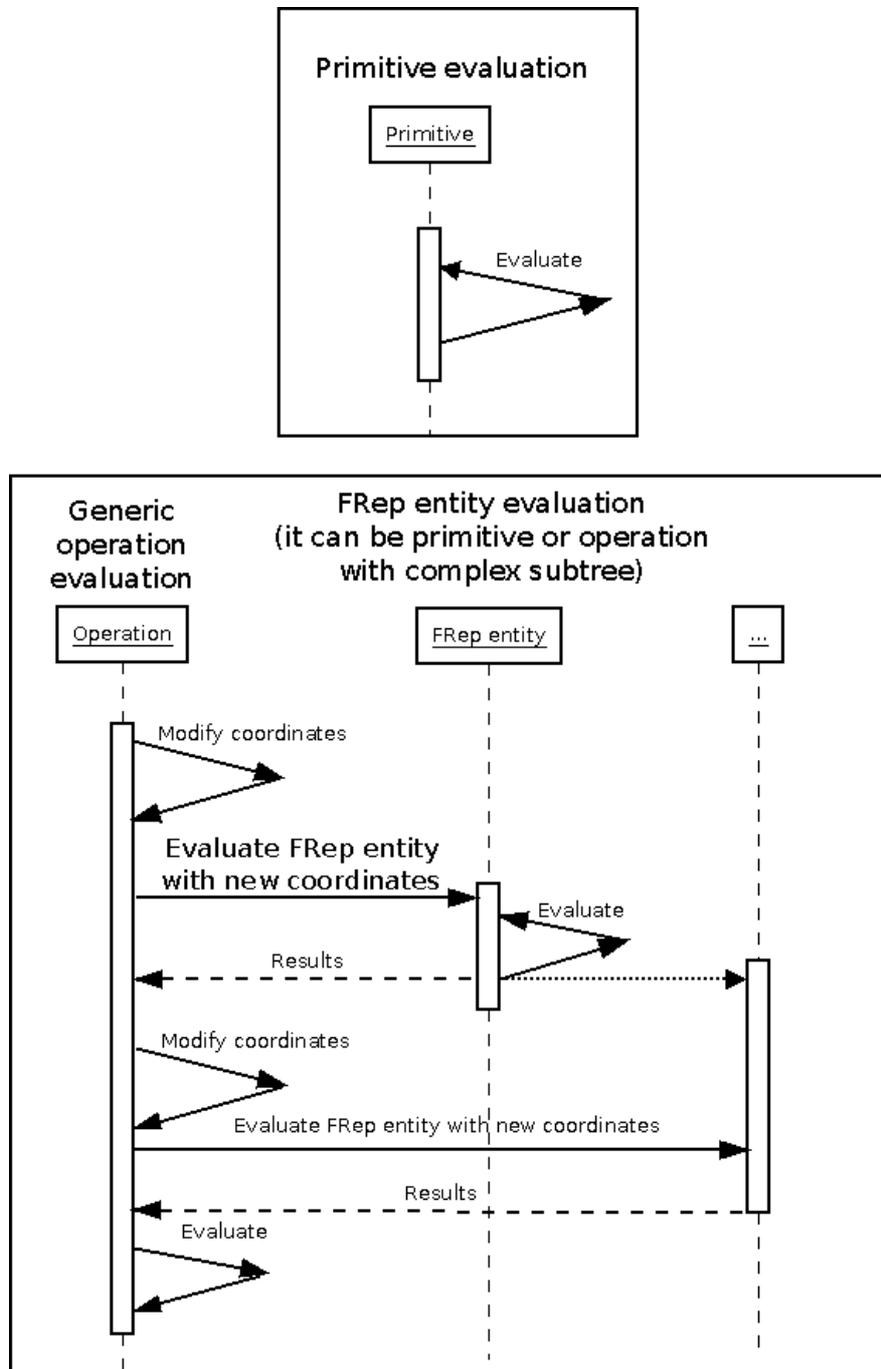


Figure 66: The dynamic diagrams illustrating the evaluation of different types of entities.

All the connections between the entities are done with the help of an additional data structure described in detail later. This approach allows us to separate the concepts of the FRep entities and the constructive FRep tree describing the model. An entity located at the root of a constructive tree implicitly initiates the evaluation of the whole tree when requesting the evaluation of the entities that it works with. These entities in turn request the evaluation of the entities that they work with. This process is repeated until primitives are requested to be evaluated. The list of available operations is shown in the fig. 64.

Attributes

The combination of geometric objects and attributes allows the user to construct Hypervolume objects. This is a crucial feature for the definition of complex heterogeneous objects. Attribute objects can be added to the tree just as any other regular FRep entity. After such integration the FRep tree is treated as a Hypervolume object. Evaluation of such an FRep tree results in a geometric object (a space partition) and a set of attributes assigned to it. A brief list of the available entities working with attributes is shown in fig. 64.

The FRep tree structure

It has already been mentioned that we aim to provide different levels of model representation. This will allow us to work with the model in terms of geometric modelling, in order to create new geometric primitives, and to combine these using operations. Equally, there should be an option to manipulate the model at a lower level in terms of the constructive tree and its nodes. Constructive tree nodes are used to establish connections between the entities (see `ENTITY_NODE_T` in fig. 67). The nodes are elements of the entire tree structure. They also perform the validation of the connections being established between entities and they ensure that non-compatible entities are not combined together. The `ENTITY_FACTORY_T` is used to create the FRep entities registered in the API (more details are provided in section 4.6.3). Additionally, the FRep tree can be easily modified on the fly through the replacement of entities stored inside its nodes. This can be used for quick modification and parameterisation of the model.

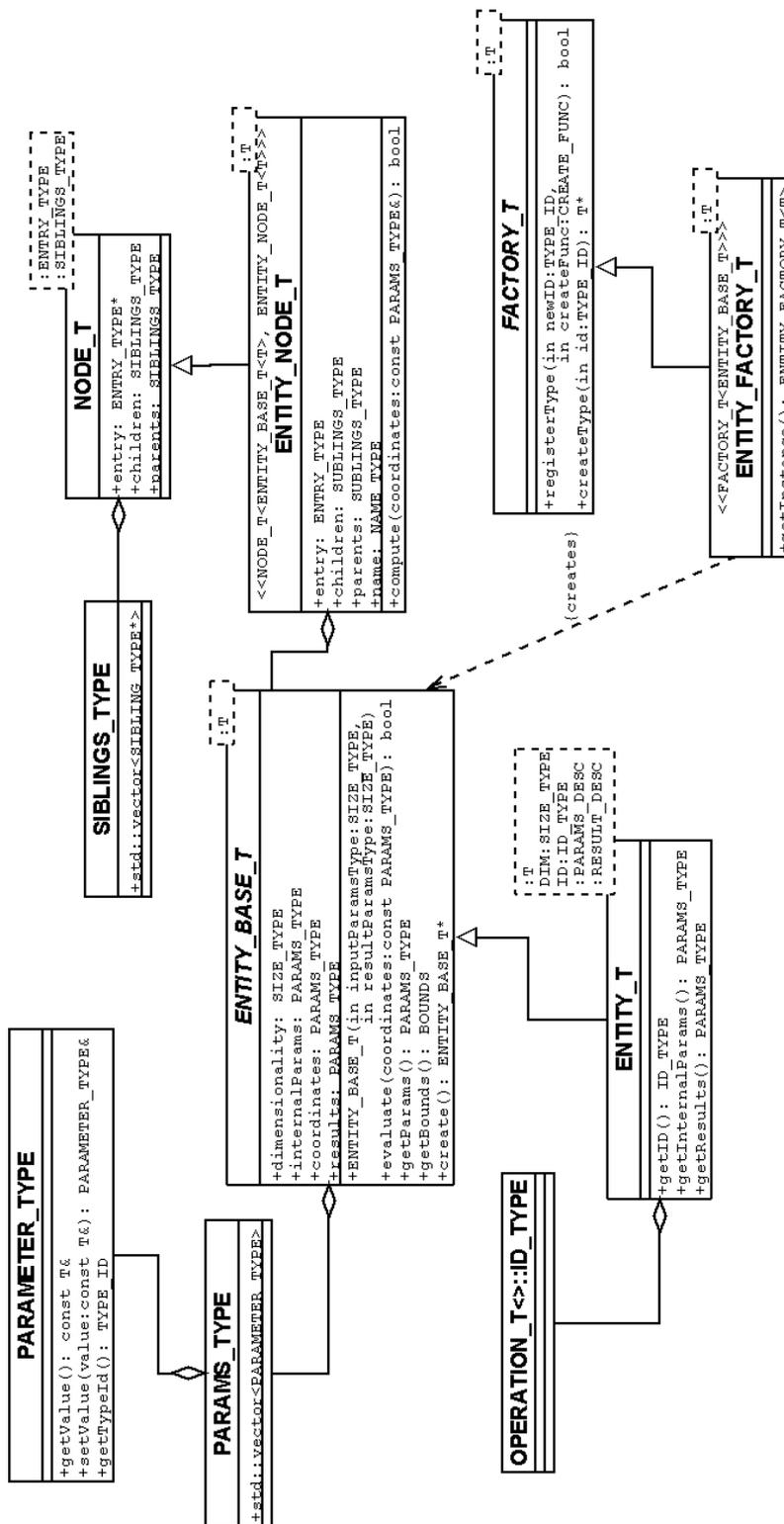


Figure 67: The FRep entity UML-diagram.

Once a tree structure has been validated, its actual evaluation can begin. Evaluation of the tree is performed in two phases. Each entity is provided with the current coordinates of a point where the function or attribute values need to be evaluated. In the first phase each entity modifies the coordinates of the point passed to it, if it is required to do so (see fig. 68). The modified coordinates are then passed to the entities contained in a node down the FRep tree. This procedure is repeated recursively until a leaf of the tree is reached. At this point the second phase of the process is initiated (see fig. 69). Each entity evaluates a value or a set of values associated with the supplied point in space and other parameters. The evaluated results are then transmitted to the entity contained at the higher levels of the tree, until the root node is reached. The result evaluated by the root node is considered to be the result of the FRep tree evaluation.

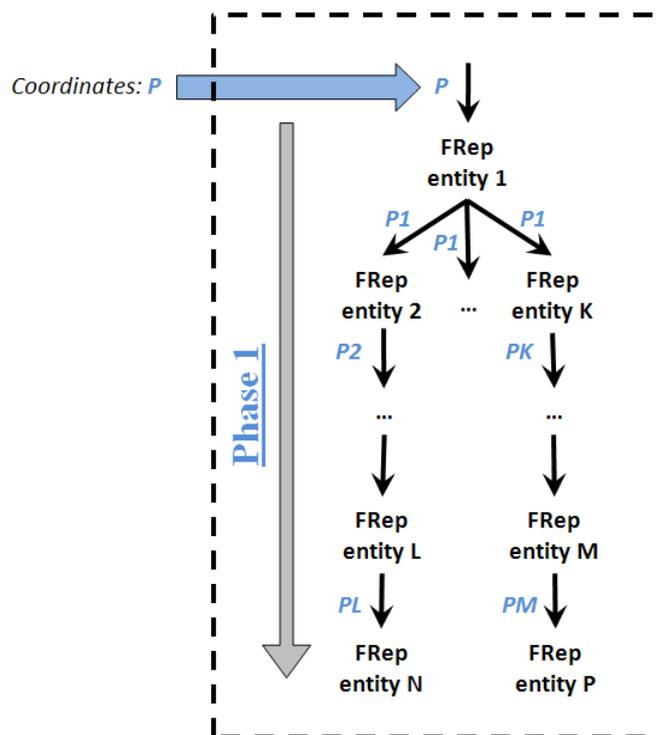


Figure 68: *The first phase of the FRep tree evaluation.*

The tree or sub-tree structure, which is built using nodes, provides additional functionality allowing us to work with an FRep tree. This functionality includes searching for particular entities or entity types, the modification of the tree structure, tree manipulations (such as copying, subtree replace-

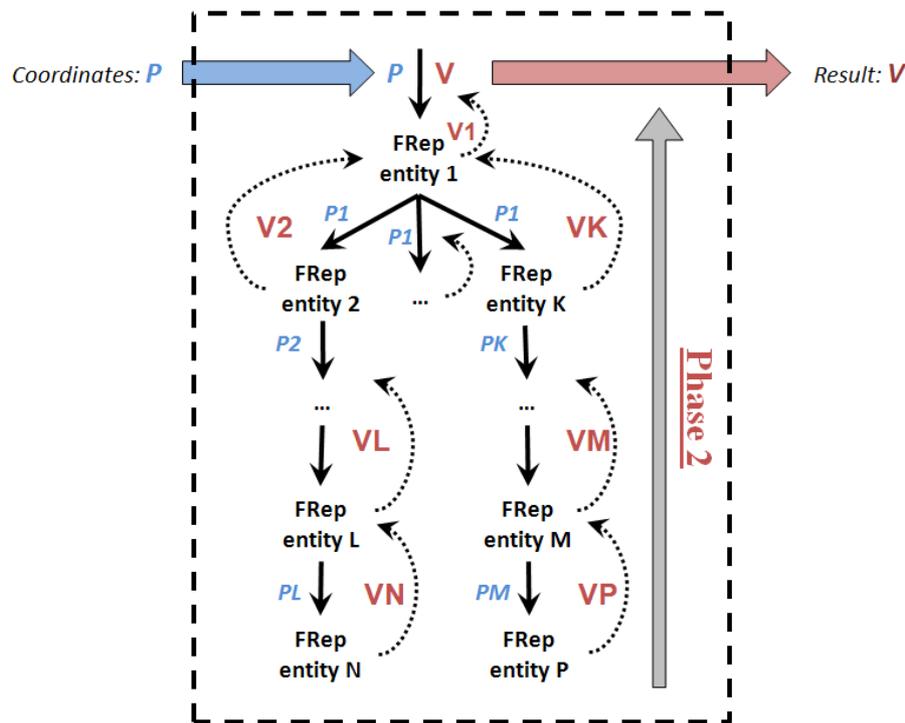


Figure 69: The second phase of the FRep tree evaluation.

ment and replications), the per-node application of user-defined algorithms and other functions.

4.6.3 FRep API extensibility

The standard interface for an FRep entity described in 4.6.2 helps unify the whole set of currently available entities as well as any new ones that could potentially be added in the future. For any new entity we will need to implement an interface `ENTITY_BASE_T` (see fig. 67). If this is done, any such new entities can be used in an FRep tree as any other entity. Hence the addition of a new FRep entity is pretty straightforward.

One of the important features we need to provide is the dynamic creation of concrete instances of FRep entities at run-time. It is also important to be able to work with diverse entities in an abstract manner; so that we do not need to access the source code of the implementation of the entity. This mechanism can be implemented with the help of a design pattern “factory method” (Gamma *et al.*, 1995). Each entity needs to be registered with a

global factory object of type `ENTITY_FACTORY_T`. Registration means that each entity has to provide its unique identifier, its internal parameter descriptions and the method which can be used to create a new instance of this entity. All the entities are later created through requests to the factory object that only needs the identifier of the entity being created. Further modifications of the entity can be performed through changes to its parameter values in a unified manner. This allows us to extend an existing set of FRep entities through their dynamic registration with a global factory object. This can be done at run-time allowing us to use the entities which were unknown at compile time (in a way similar to plug-ins in some software packages). New entities developed by third parties can be added without any modification or recompilation of the core FRep source code. This conforms to the concept that each FRep entity should be thought of as a black box with a common set of inputs and outputs.

4.6.4 FRep model manipulation

One of the essential goals of the FRep API is to provide the user with a software framework which can be employed by a number of external applications depending on their specific needs (see fig. 70). It should be possible to integrate the produced FRep models into a hybrid model defined within an IC framework or to use the resulting FRep models independently. This also implies that there should be different ways to define the actual FRep model using the underlying features provided by the FRep API.

An FRep model could be created and manipulated directly using a high-level programming language. On the other hand the functionality provided by the API could also be exposed to a scripting language. Herewith the user is given the ability to define a model at a higher level of abstraction without the need to recompile the code in order to see the results of the model evaluation. Support for one of the widely-used scripting languages can be provided in a relatively simple way. For instance, all of the registered FRep entities can be exposed to Python (Python Software Foundation, 2009) with the help of the Boost.Python library (Abrahams and Grosse-Kunstleve, 2009). It might be easier for users unfamiliar with high-level programming

languages to define FRep models using such a popular interpreted language. Another advantage of this approach is the fact that modification of the source code written in Python does not require recompilation. Obviously, compared to compiled C++ code, poorer performance of the model evaluation will be achieved. This is acceptable at the proof of concept stage. Alternative scripting languages could be used including a custom domain specific language built around FReps. This could be a language similar to **HyperFun** or **HyperJazz** (Adzhiev *et al.*, 1996).

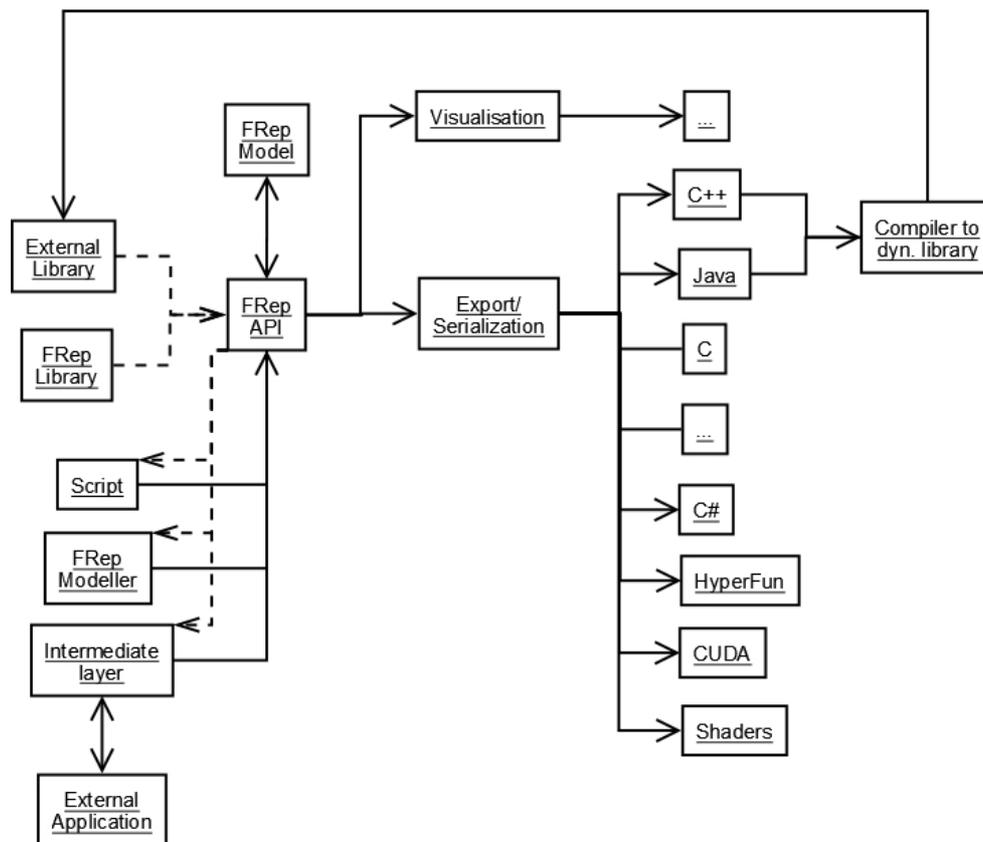


Figure 70: *The interaction with the FRep API.*

Another way to define an FRep model is to provide a user with a graphical user interface (GUI). According to a “*Model-View-Controller*” design pattern (Gamma *et al.*, 1995) (see fig. 71a) can be introduced to the model (in this case the FRep model) via a controller (in this context a layer performing the processing the user input and mapping it to calls of the FRep API). The current state of the model is reflected in a view. This can be a GUI or specific textual information reflecting particular properties of the FRep model.

This is the standard method of de-coupling the specific model domain layer from the presentation layer. The separation of responsibilities between the model between the view and the controller allows us to concentrate on each task independently and to change existing components or to introduce new components to the system more easily.

A custom GUI built around an FRep API could be designed to provide a way to create and to manipulate the FRep model (fig. 71b). A generic GUI of existing third party software packages could also be used to manipulate the FRep model. In this case we need to provide an intermediate layer between the FRep API and the specific API of the modelling application we wish to extend. We provide a concrete example of the FRep API integration into a third party modelling software package, namely Autodesk® Maya™, in section 5.7.

Different ways of FRep model manipulation are reflected in figure 70.

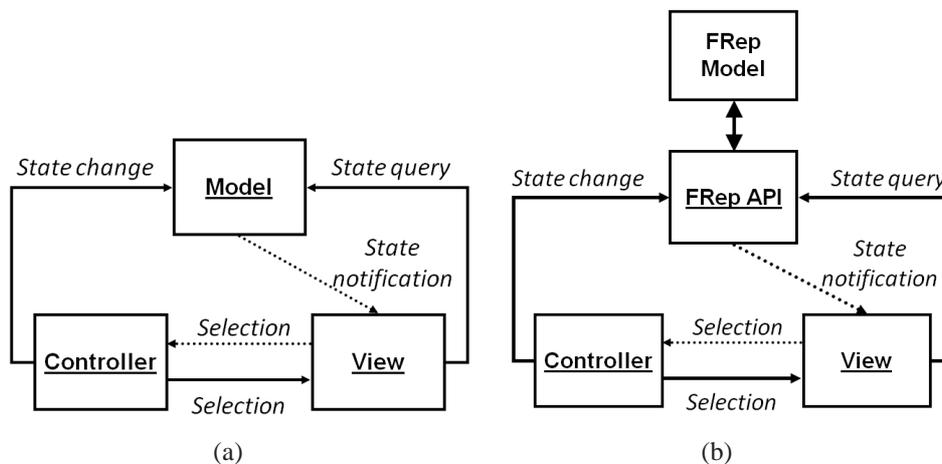


Figure 71: A Model-View-Controller diagram: (a) The general design pattern (b) The MVC and FRep API

4.6.5 The FRep model format interchange

The Function representation is a universal model representation which can be implemented in different programming languages. In specific cases we might wish to convert a particular FRep model from its internal representation to an FRep model described in an alternative fashion. This is important in order

to allow the user to exchange FRep models between different modelling environments and available programming languages. This might be beneficial from a performance point of view or it might be required for a specific application area. Thus, we need to provide a simple way that allows us to store an FRep model, which can then be mapped to the required representation.

In order to achieve this we can borrow a technique from computer science called serialisation. Serialisation is a technique used to convert a computing object into a set of data, which can then be used to reconstruct the original object, so that the object can be stored into a storage medium for later retrieval. The piece of program code that performs this serialisation is called the serialiser. In our framework, the serialiser can be thought of as the component that is responsible for the serialisation of an abstract FRep entity. The serialiser is aware of the particular type of an FRep entity and its parameters and it provides the functionality that converts this information to an appropriate representation (fig. 72). The serialiser manager retrieves information concerning the internal connections between various FRep entities and the global properties of the model and maps this information to an appropriate representation.

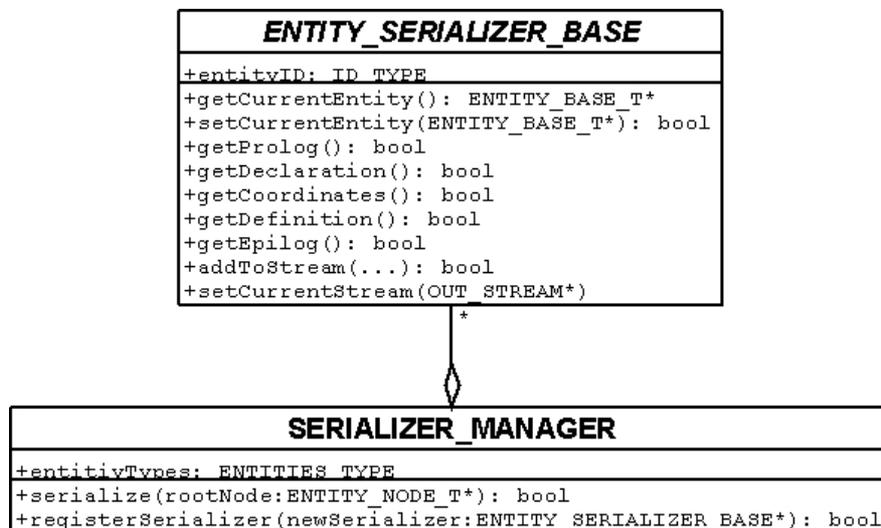


Figure 72: *The entity serialiser.*

A number of serialiser managers can be registered. Each such manager is responsible for the conversion to a specific representation. It is important to note that the FRep API itself has no information about other possible repre-

sentations. It only provides a common way to get access to the information regarding the entities and the structure of the model, which in turn can be converted into another representation by a set of concrete serialisers. The serialisers are registered in a way similar to the entity registration. Each serialiser is required to provide the unique type identifier of an entity that it can serialise. During the process of the FRep model conversion, the methods of this serialiser will be invoked and it will be provided with an entity which requires conversion. A set of serialisers and serialise managers can be extended at any time to support the ability to export a description of the FRep models to other specific representations.

One interesting application of run-time FRep model serialisation is related to model compilation. The description of an FRep model can be exported to one of the languages allowing compilation of the code to a dynamically linked library. In this instance the entire model can be serialised and compiled for efficient evaluation. The compiled model can then be registered as a single FRep entity at run-time and be used in the FRep model instead of a complex constructive tree. In some cases this could lead to a performance boost, since an advanced compiler would produce native code which is highly optimised for a specific target platform. Fig. 70 illustrates this idea.

The serialisation approach is also a natural way to convert the internal representation of FRep models to hardware specific descriptions. For instance, a constructive tree consisting of FRep primitives and operations can be directly converted to a programmable hardware shader that can be used to perform fast ray-tracing on the GPU (Fryazinov and Pasko, 2008). Such a conversion and shader compilation can happen on the fly, which would allow the user to see the model he or she is working on in real-time (or near-real time depending on the model complexity).

We also need a unified way of model representation in order to allow various applications to exchange the produced models. This should be a lightweight format, which can easily be loaded and saved. It is preferable to take advantage of a human readable format, to allow the users make manual modifications of the model. We have chosen the Extensible Markup Language (XML) (W3C, 2010) as our main format for FRep model interchange.

This is a widespread format which can be used for storage of arbitrary data. Working with XML can be done using a large set of libraries available for different programming languages and hardware platforms. An FRep tree is saved in a hierarchical structure. Parameters of the entities are saved as XML-attributes of different data types. An exemplar FRep model and its serialised definition are shown in fig. 73.



Figure 73: The FRep entity and its XML definition.

FRep models saved in FRep XML format can be directly loaded by the dynamic IC framework through the IC API. Such integration of FRep models into an IC API simplifies the way of handling the F-cells within the IC framework and makes the process of integration of FReps into a hybrid model rather trivial.

4.6.6 The FRep library

In section 4.6.3 we described the requirements for the extension of the FRep API. This is a straightforward process, which requires only a basic knowledge of the FRep API. Apart from adding the entities to the API we need to provide a way for them to be serialised in order to be able to save and restore the state of the FRep model (see section 4.6.5). Certain types of action required the addition of the FRep entities to the API and their serialisation could be automated, thus allowing us to avoid repetitive work. In this section we will describe the approach that allows us to achieve this.

One of the essential goals of the FRep API is to provide the user with a software framework which can be employed by a number of external applications depending on their specific needs (see fig. 70). The main application we are concerned with is the dynamic IC framework. But we need to consider a way of simplifying the integration of the FRep API into other applications as well. The main and most common way of communication between the FRep API and higher-level applications using FReps is through the parameters of the FRep entities and through the creation of complex FRep trees containing the aforementioned entities. As was mentioned before there is often a necessity for the presence of an intermediate level of the application, which performs the mapping of the FRep entities to application specific terms. Here we propose an approach which allows us to simplify the process of intermediate layer generation.

Each entity encapsulates a predefined set of its characteristic properties which include¹⁶:

1. **Internal parameters.** These parameters are a primary way of manipulating the entity and adjusting the resulting shape or its volumetric attributes.
2. **A set of input parameters.** Input parameters are required by FRep operations in order to combine the function values of the operands. This parameter set is empty for FRep primitives.

¹⁶Each parameter and procedure may have its counterpart required for the evaluation of function interval. Functions performing interval evaluation may be specialised depending on the interval estimation method or the specific mathematical functions being used.

3. **A set of temporary parameters.** Temporary parameters are normally those that depend on the internal parameters and need to be re-evaluated only when the internal parameters they depend upon are modified.
4. **The main evaluation procedure.** In this procedure the resulting function value is evaluated. Each FRep entity provides this procedure in order to perform a mapping of the coordinates from the modelling domain to a scalar or a vector value.
5. **The coordinate modification procedure.** This procedure performs the space-mapping. This is required for the first phase of the FRep tree evaluation (see fig. 68).
6. **The operand-dependent coordinate modification procedure.** This procedure is similar to previous one. The main difference being them is the fact that this procedure needs a value provided by its operand(s) in order to modify the coordinates of a point (for instance, it may be used for shape-driven deformations (Schmitt *et al.*, 2003)).
7. **The temporary parameters evaluation procedure.** A procedure describing how temporary parameters should be evaluated.

This description can be provided in a human readable format, which is then parsed and transformed into an Abstract Syntax Tree (AST) or even into an Abstract Semantic Graph (ASG). Having a valid description of the entity in this format, we can produce the code necessary for the implementation of FRep API for different software and hardware platforms. In order to do so, we need to provide a specific translator which traverses the AST and converts it into the desired format. Figure 74 illustrates how entities for the C++ FRep API can be added. This is useful for the initial code generation where a certain amount of manual work is required. But more importantly, this simplifies the maintenance of the APIs across different platforms ensuring that they are all consistent. Otherwise any modification introduced to any entity requires a careful manual update of all the code-base.

Apart from reducing the burden of the manual work required for each FRep entity, we also simplify the maintenance of third party applications over time. As we mentioned in section 4.6.4, an intermediate layer is required to bridge

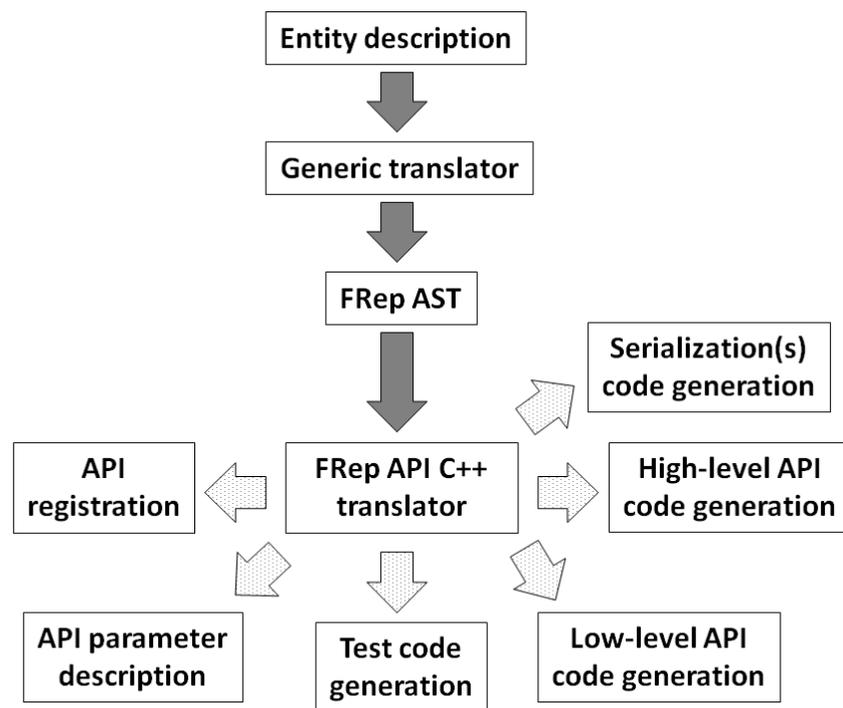


Figure 74: The generation of the FRep API related components from an entity description.

the gap between the FRep API and the third party application. This bridging process can in fact be rather tedious and time-consuming. This applies to situations where new entities are added or where existing entities are modified in any way¹⁷. To simplify the integration process an application specific translator needs to be created. This translator does not necessarily need to generate the code for the actual implementation of the FRep entities, if the application can rely on the FRep API in existence for a particular platform. The translator could take advantage of the aforementioned descriptions in order to generate the necessary intermediate code required for the smooth integration of the FRep functionality. For instance, to reflect parameters of the entities and to retrieve or modify states of the entities using application specific methods (see fig. 75). This way the FRep API could be integrated in a number of applications in a relatively easy way, through a set of translators aware of the application specific needs (see fig. 76). Any modification in the description of any entity would be automatically reflected in all the dependent

¹⁷This may be required when new features are added to existing entities, in order to resolve discovered issues or in other situations that we might not be aware of at the time of implementation.

applications through an automated translation process. This way, keeping all the applications in sync with the up-to-date state of the FRep API should be rather straightforward.

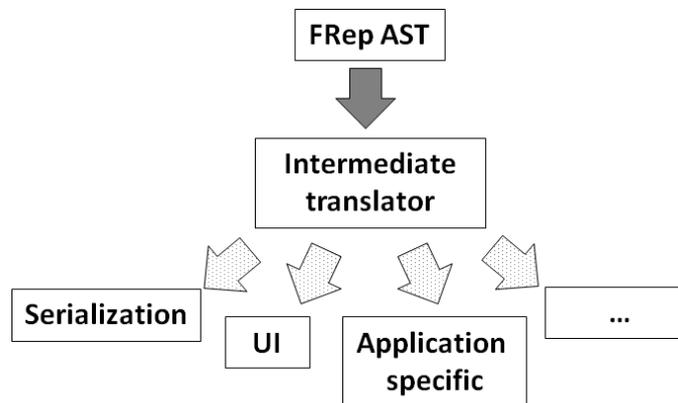


Figure 75: The generation of application specific code required for the integration of FReps.

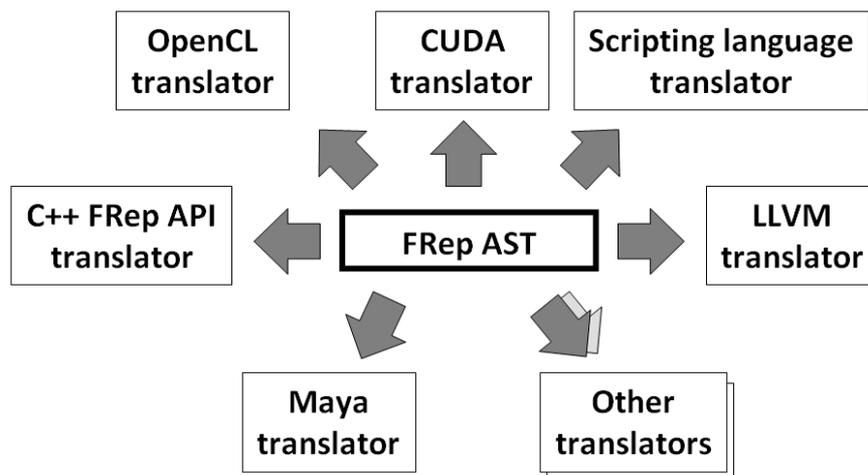


Figure 76: A set of application and platform specific translators.

4.6.7 Integration of FReps into the IC framework

It is important to provide the users with the appropriate tools required for the definition of an FRep model which can be integrated into a hybrid model. Depending on the specific needs of the users a different set of modelling tools may be more suitable for the definition of the actual FRep model. The resulting model can be exported from any of these applications using a common

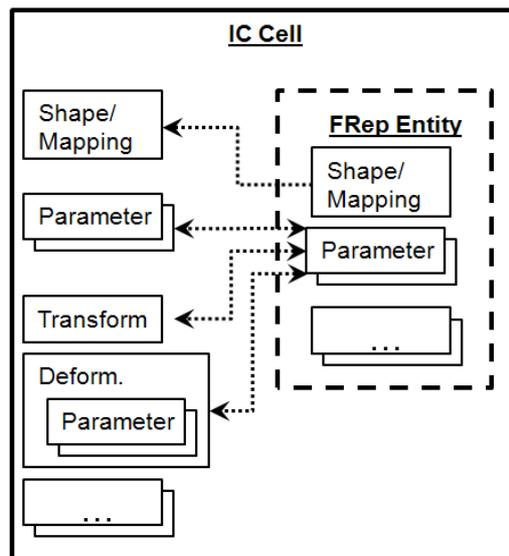


Figure 77: *The integration of the FRep entities into the IC API.*

storage format (see section 4.6.5). The exported FRep model can then be integrated into a hybrid model as an F-cell or its attributes, since the IC API relies on the FRep API to provide the functionality required for handling these models. IC entities represented by FRep cells or attributes can expose these parameters and subtrees to the hybrid model, making it possible to change their state using the same set of tools as for any other IC entity (see figure 77), provided that the parameters and other properties of the IC cells can be used for the evaluation of the shape or of the mapping defined by an FRep entity contained in a an F-cell. In this case the resolution of all the dependencies and dynamic modification of the properties of the F-cells are performed in the same way as for all other IC entities (see figure 78). The discretisation engine allows us to provide a polyhedral representation of any F-cell as required by the IC framework.

The flexibility of the FRep API allows us to work with standalone function representation models and to easily integrate them into complex hybrid models. F-cells and attributes integrated into our dynamic IC framework greatly enhance our ability to model time-variant multi-dimensional heterogeneous objects.

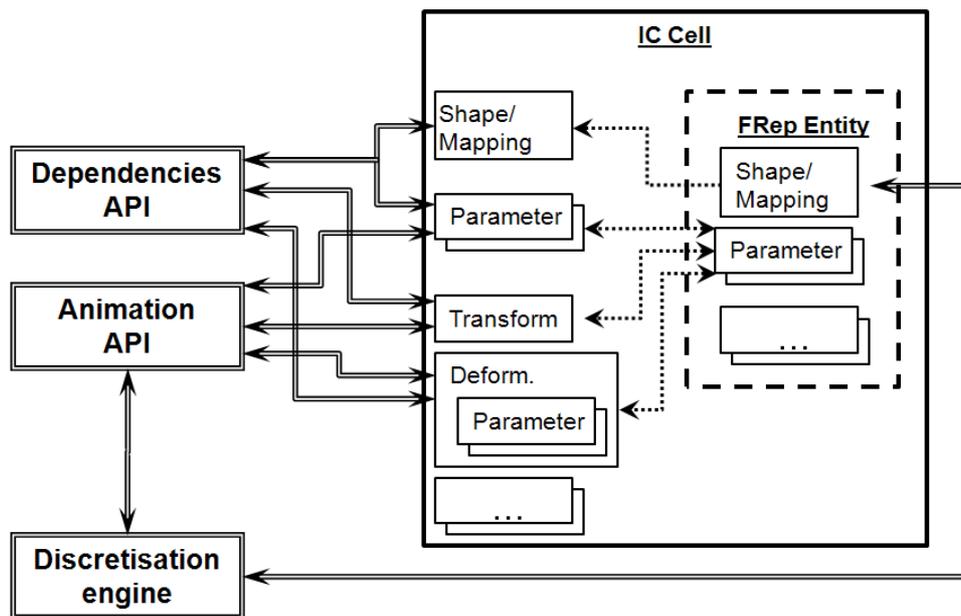


Figure 78: The evaluation of the F-cell within the IC framework.

4.6.8 Performance

One of the most significant shortcomings of FRep models is their computationally expensive model evaluation procedure. Here we will discuss a number of ways to alleviate this problem.

LLVM compilation

We have already outlined a number of advantages of the FRep API which allows us to flexibly define mixed-dimensional models of varying complexity. But the flexibility of the API also results in an abstraction penalty, i.e. when we compose complex models and have the ability to modify them on the fly, we establish and set up a set of internal structures which need to be traversed at run-time. The discretisation system then has to traverse these structures and to perform heavy computations. Once these structures are set up, we can convert them to a more efficient representation that provides us with less flexibility but can be evaluated faster. Building a single monolithic FRep function could help us greatly enhance performance. This can be done using the approach described in section 4.6.6. Figure 76 illustrates the possible conversion of an FRep AST to a program defined in terms of LLVM bytecode. The LLVM definition of the entities can then be converted to the native

code of various hardware platforms taking into consideration their specifics and using them for the optimisations. Target platforms could include both CPUs and GPUs.

Parallel execution

FRep models are well suited for parallel evaluation. The modelling space can be divided into a number of non-intersecting subsets. Each of the subsets can thereafter be evaluated independently. FRep entities store some specific data needed during the tree traversal procedure (for instance, the modified coordinates of a point in modelling space are propagated down the tree). If we wish to evaluate these entities in different threads, collisions might occur, that will lead to invalid results of the model evaluation. It is thus preferable to provide each thread with a separate copy of the same FRep tree in order to avoid multi-threading issues. Parallel evaluation of the model can be performed on one machine. Another possible way to evaluate a model is to convert it to a hardware specific representation and to employ one of the existing General-Purpose computations on the GPU (GPGPU) methods. We can export the internal model representation to an NVIDIA CUDA kernel (NVIDIA, 2010), which would subsequently be compiled directly into GPU native code relying on the CUDA FRep library. This CUDA-specific FRep library can be generated using our generic approach described in section 4.6.6. The kernel produced after export would be executed on hundreds of GPU ALUs in parallel. Two approaches could be used:

1. The application could export the model that could be processed and translated into specialised data sets and CUDA kernels, which are executed on the GPU. This can be easily implemented through a custom serialiser as described in 4.6.5. This serialiser simply traverses the FRep tree and exports it as a set of calls to FRep library implemented through CUDA. We have provided a more detailed description of the application of this approach in (Pasko *et al.*, 2010).
2. The application could generate input data sets (parameters of entities), which could be fed as input data to an existing generic kernel. This kernel would traverse the constructive tree, calling appropriate functions with parameters extracted from the provided data. This could be less

efficient compared to the former approach in terms of execution time and memory usage, but a description of relatively complex segmented models could still fit in shared memory or be cached in read-only memory.

Further research is needed to determine which of these methods is best suited for particular situations. However it is already clear that the compact representation of FRep models is also well suited for execution on a GPU, which has a limited amount of shared memory available for each kernel. Results of the model evaluation on the GPU can be fed back to RAM or used to render the model on the GPU (see 4.6.8). Alternatively, the evaluation of complex FRep models can be distributed across different computers on a network. In such a case the “server” would store the initial FRep model and would send its description to a number of “client” machines. The “clients” would evaluate parts of the model and send the results back to the “server”, which in turn would arrange all the retrieved results together and return them to the user. If all the “clients” are ran on the same operating system, the “server” could send a compiled dynamically-linked library containing the model, which is compiled into native code, instead of a model description. In this case, the entity stored in the dynamic library could be registered by all the “clients” and evaluated as any other generic entity.

Tree pruning

Complex FRep models are represented by constructive trees of significant depth. This means that the evaluation of such models is computationally expensive. It is apparent that for some of the models the majority of the FRep entities do not intersect. Distinct entities occupy specific subsets of the modelling space but do not exert any influence in distant regions of this space (i.e. a scalar field produced by such an entity does not contribute to the regions outside the specific bounding volume of such an entity). Tree pruning (Fox *et al.*, 2001) is a method that lets us “simplify” constructive trees for particular subsets of the modelling space. Tree pruning can create a set of FRep trees characterised by a lesser depth compared to the tree describing the entire model. Each tree approximates the model for a given subset of the modelling space. If we wish to exploit this approach, each FRep entity is

required to provide a method allowing us to determine its bounding volume. The calculation of the bounding volume is simple for a number of known primitives but can be a complex task for non-trivial operations (such as for blending or for non-linear deformations). Thus, further research in this area is required. This technique might be especially useful for complex models such as those of virtual environments where the user is given the freedom to move around and to modify the model of the scene.

Discretisation optimisations

Discretisation of the model can be performed using one of the existing polygonisation methods (Lorensen and Cline, 1987; Bloomenthal, 1994; Hilton and Stoddart, 1996). Polygonisation allows us to retrieve a polygonal approximation of a particular iso-level of an FRep model. The quality of the resulting approximation depends on the size of the step chosen for the discretisation. Standard techniques used to control the quality of the model approximation, based on some heuristic (Clark, 1976), can also be employed for an FRep model. For instance, a geometric object located in the subset of space within a given proximity to the viewer needs to be approximated with a higher precision than an object located further away from the viewer. Such view-dependent techniques can significantly decrease the evaluation time required to visualise an FRep model (Kazakov *et al.*, 2001). Special care needs to be taken at the boundaries of the regions discretized with different precision, otherwise cracks between such regions may appear. Additional optimisations can be introduced when polygonising dynamic FRep models, in which case we need to track changes to the model that have taken place after the previous polygonisation stage. This implies that we need to track the transformations applied to all the entities over time and to perform a partial polygonisation of the regions affected by these changes. In the more general case, re-polygonisation only needs to be performed in the proximity of the areas where the surface was tracked in previous frames. Additional attention must be paid to the objects appearing in the scene at specific moments in time. A higher frame-rate during the visualisation of a dynamic FRep model can be achieved if the polygonisation is not performed for each frame. Iso-surfaces can be extracted at predefined moments in time (e.g. only on each third or fourth frame). In such a situation, the intermediate visual represen-

tation of the model can be constructed through traditional alpha-blending of iso-surfaces extracted at adjacent moments in time. In any case, employing time-coherence for dynamic models can lead to a significant performance boost. If only static models are present in the environment, re-polygonisation only needs to be performed in the areas of local change introduced by the user.

Direct rendering

Polygonisation is not the only method available for the visualisation of FRep models. Real-time or near real-time ray casting of the model can be performed on the GPU (Fryazinov and Pasko, 2008). This requires the conversion of an FRep model to a hardware specific representation (as mentioned in 4.6.5). The FRep API provides a relatively easy way to perform a conversion from the internal representation to one of the existing shading languages or to a set of CUDA kernels (see 4.6.8). This means that not only a specific iso-level of the model can be rendered but also a full-blown GPU volume rendering technique can be utilised. Another option would be the voxelisation of the model in the camera frustum.

4.7 Conclusions

In this chapter we have presented a detailed description of the implementation aspects of the new dynamic IC framework. A set of components constituting the framework were defined and described in detail. We have also considered issues related to the evaluation of a dynamic hybrid model. Along with all the aforementioned aspects of the IC framework we have introduced a high-level notation, which can be used for the definition of actual IC-based models. Using this notation the user can provide a detailed description of the model and request its evaluation using the IC framework. We have paid particular attention to the FRep components of the IC framework because of the high importance of this representation for the definition of heterogeneous objects and because of the limitations of the existing tools required for FRep modelling. We proposed a methodology for the definition of a multi-platform software framework for FRep modelling, which can be implemented

in a number of ways. We described our implementation of the FRep API, thus making FRep more accessible to a wider group of users. We provided an overview of the methods which can be used for the acceleration of the evaluation of the aforementioned models. Apart from the main description of our newly developed software tools we have outlined possible directions for future developments. The provided description of the IC framework and its FRep components should be sufficient for further implementation and improvements.

In the next chapter we will provide a set of detailed case studies. These case studies demonstrate how the proposed dynamic IC framework can be used in a number of different applications in order to solve a set of existing problems.

5 Applications and results

In this chapter, we describe a number of applications of the proposed modelling framework and discuss some results. The problems under discussion are formulated in terms of dynamic hybrid models, which allows us to take advantage of the expressiveness and power of the IC framework. We outline a set of improvements proposed for space-time blending relying on multidimensional dynamic models in section 5.3. Next we introduce our new method for the modelling of interactions between dynamic objects and viscous materials using time-dependent hybrid models (see section 5.4). We also describe the extensions to this approach which allow us to solve a number of other existing problems, including partial metamorphosis of animated characters and the controlled metamorphosis of dynamic meshes presented in section 5.5. Then we present an application involving a complex interaction sequence between a set of interdependent time-variant heterogeneous multidimensional objects within one hybrid model (see section 5.6). Finally, we describe our prototype implementation of an interactive modelling system in section 5.7. This system can be used for the definition of dynamic heterogeneous objects and certain parts of a hybrid model. In addition, we describe in detail practical methods that can be employed to accelerate model evaluation employing both the CPU and the GPU.

Overall, we demonstrate that heterogeneous object modelling is a powerful way of overcoming a set of existing problems, some of which are next to impossible to solve using other existing methods.

5.1 An introductory 2D dynamic IC model exemplar

We start with a simple example before proceeding to more advanced problems. This example will help demonstrate what is involved in generating a detailed and complete description of even a very simple hybrid model. For this example we will produce a full model definition as described in previous chapters.

The simple model consists of two geometric entities: a disk and a rectangle

(see fig. 79).

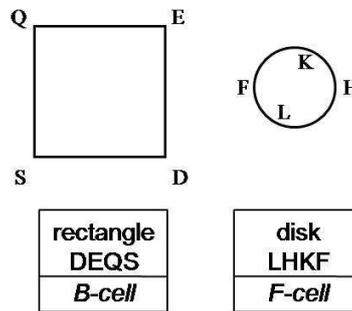


Figure 79: A set of cells present in the IC.

The disk LHKF moves over time. According to the definition of the cell in section 3.2, we need to provide a set of parameters values and to define the way these parameters will change over time. The translation transformation of the disk $LHKF$ is defined by the set of parameters:

$$p_{LHKF}(t) = p^0_{LHKF} + v_{LHKF} \cdot t$$

where p^0_{LHKF} is the initial position of the centre of the disk (which is defined using $M_i(t)$) and v_{LHKF} is the velocity of the disk ¹⁸ (see fig. 80). In this example we assume that the rectangle $DEQS$ is a static cell with its centre at the point p^0_{DEQS} . (i.e. its $M_i(t)$ is constant over time). The shape, of the rectangle $DEQS$ is defined using an FRep (see section 2.2.3), while disk $LHKF$ is defined using a BRep (see section 2.1.2). In this simple model we integrate two objects expressed in two different representations.

We provide a textual definition of this dynamic IC model using the notation introduced in sections 4.3.1 and 4.3.2. According to the methodological recommendations introduced in section 4.3, we start our description of the model with the definition of the cells contained within it ¹⁹:

```
// description of persistent cells
CELLS {
  DEQS {
    type = B-CELL;
    // use one of the library objects setting up appropriate parameters
```

¹⁸The velocity could also be defined as a set of control points and time values. In general we need to provide a number of different ways for defining the transformations and all other parameters.

¹⁹No custom events will be used for the definition of this simple model.

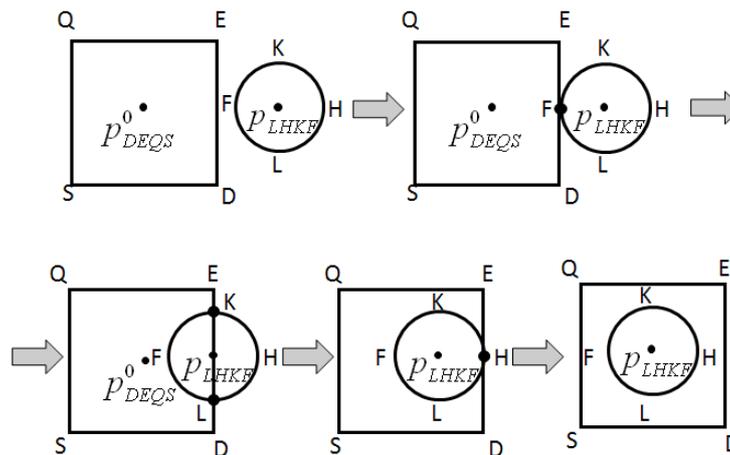


Figure 80: *The motion of the disk defined over time.*

```

shape = library::rectangle(...);
dim = 2D;
domain = {...};
// we will only be using built-in transforms, thus no additional
// parameters will be defined
parameters {
  translation(...); // define initial translation
}
} // DEQS
LHKF {
  type = F-CELL;
  // load the description of the FRep tree (XML, HyperFun etc)
  shape = frep::loadTree(...);
  dim = 2D;
  domain = {...};
  // custom parameters of the cell
  parameters {
    translation(...); // define initial position
    velocity : REAL3(...); // how fast the object is moving
  }
  // reactions to certain events:
  REACTIONS {
    // define simple motion over time:
    update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
    {
      // update position:
      translate.relative = globalT * velocity;
    }
  }
} // LHKF
:
} // CELLS

```

In order to gain a better understanding of the entire model we need to

introduce additional parameters for the definition of the parametric state of the dynamic IC model (see section 3.7). In this example we will use a parameter reflecting the state of the IC:

```
STATE_PARAMETERS {
  // parameter reflecting whether the cells have intersected
  isIntersection : BOOL( false );
}
```

The value of the parameter is set by the active IC instance.

From figure 80 we can see that the model has at least five different structural states or IC instances²⁰ (see section 3.7). Each structural state is reflected by an IC instance. For simplicity we only provide descriptions of the first three IC instances. The last two IC instances are derived from the previous IC instances through the addition of the containment relations.

To start with, we define an IC template that is equivalent to the initial IC instance (fig. 81). This is done using the syntax described in sections 4.3.1 and 4.3.2.

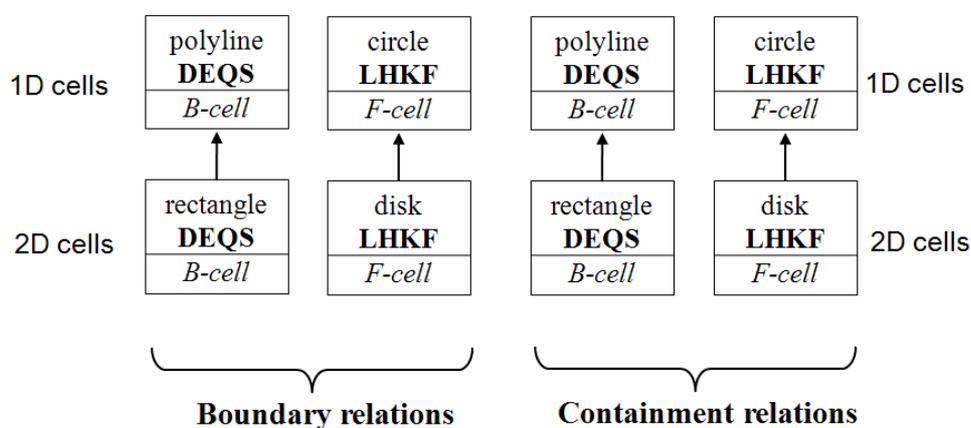


Figure 81: *The topological relations of the IC.*

```
K1 {
  // references to a set of previously described cells
  CELLS {
    // copy all the cells from a template IC
    USE INITIAL_TEMPLATE.CELLS;
  }
  // references to a set of previously described cells
```

²⁰The last two states are similar to the first two, excluding the additional containment relations.

```

RELATIONS {
    // we will use implicit relations from the cells,
    // no additional relations will be required
}
// events processed by the instance
REACTIONS {
    // none at this stage
}
// predicate used to find out if the instance is still valid:
PREDICATE {
    bool evaluate( REAL globalT, REAL localT, REAL dt)
    {
        // try finding the intersection between the disk
        // and the rectangle
        BOOL isIntersection = DEQS.domain.isIntersection(
            LHKF.shape.global);
        return !isIntersection;
    }
} // K1

```

The second instance of the IC (K2) is shown in fig. 82.

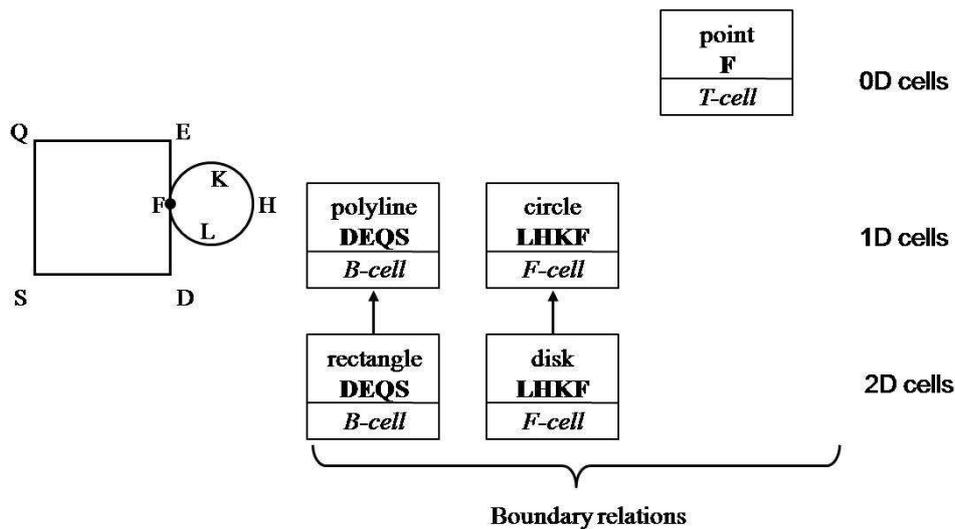


Figure 82: *The second IC instance.*

In this IC instance a new T-cell F is introduced:

```

F_point {
    type = T-CELL;
    dim = 0D;
    domain = {...};
    // the translation of this cell can be determined only after
    // the intersection test, this will be done in the instance
}

```

The second IC instance is defined as follows:

```

K2 {

```

```

// references to a set of previously described cells
CELLS {
  // copy all the cells from a template IC
  USE INITIAL_TEMPLATE.CELLS;
  F_point;
}
// events processed by the instance:
REACTIONS {
  // update position of the T-cell (Post suffix as we want it
  // to be issued after all the cells have been updated)
  updatePost( REAL globalT, REAL localT, REAL lifeT, REAL dt)
  {
    // try finding the intersection between the disk
    // and the rectangle
    INTERSECTION_INFO info = DEQS.domain.intersect(LHKF.shape.global);
    // only one intersection point exists
    if (info.intersectPoints.size == 1) {
      // set position of the T-cell using intersection info.
      F_point.translation = info.intersectionPoint(0);
      isIntersection = true;
    } else {
      isIntersection = false;
    }
  }
}
// predicate used to find out if the instance is still valid:
PREDICATE {
  bool evaluate( REAL globalT, REAL localT, REAL dt)
  {
    return !isIntersection;
  }
} // K2

```

The third IC instance (K3) is shown in fig. 83.

```

KFL_half_disk {
  type = T-CELL;
  dim = 2D;
  domain = {...};
  //the shape of this cell can be determined only after the
  // intersection test, this will be done in the instance
}
KFL_arc {
  type = T-CELL;
  dim = 1D;
  domain = {...};
  // the shape of this cell can be determined only after the
  // intersection test, this will be done in the instance
}
KL_segment {
  type = T-CELL;
  dim = 1D;
  domain = {...};
  // the shape of this cell can be determined only after the
  // intersection test, this will be done in the instance
}

```

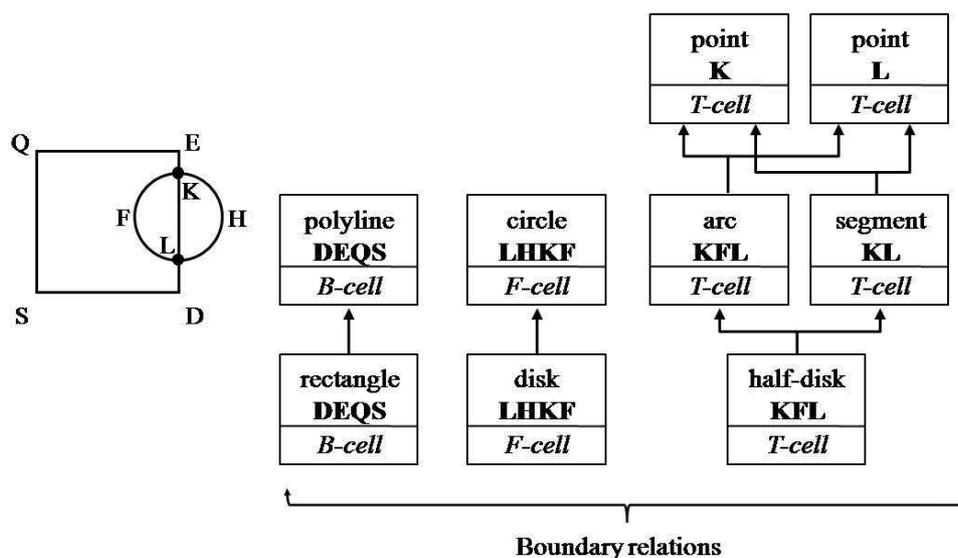


Figure 83: The third IC instance.

```

K_point {
  type = T-CELL;
  dim = 0D;
  domain = {...};
  // the translation of this cell can be determined only after
  // the intersection test, this will be done in the instance
}

L_point {
  type = T-CELL;
  dim = 0D;
  domain = {...};
  // the translation of this cell can be determined only after
  // the intersection test, this will be done in the instance
}

```

Next comes the definition of this IC instance:

```

K3 {
  // references to a set of earlier described cells
  CELLS {
    // copy all the cells from a template complex
    USE INITIAL_TEMPLATE.CELLS;
    KFL_half_disk KLF_arc KL_segment K_point L_point;
  }
  // events processed by the instance:
  REACTIONS {
    // update position/shape of T-cells (Post suffix as we want
    // it to be issued after all the cells have been updated)
    updatePost( REAL globalT, REAL localT, REAL lifeT, REAL dt)
    {
      // try finding the intersection between the disk and
      // the rectangle
      INTERSECTION_INFO info = DEQS.determined.intersect(
        LHKF.shape.global
      );
    }
  }
}

```

```

// only two intersection points exist
if (info.intersectPoints.size == 2) {
  // set position of the T-cell using intersection info.
  K_point.translation = info.intersectionPoint(0);
  L_point.translation = info.intersectionPoint(1);
  // find local centre of the intersection
  VECTOR2 center = (K_point.translation +
                    L_point.translation) / 2.0;
  // now define local shape of the segment using
  // two points:
  KL_segment.shape = SEGMENT( K_point.translation - center,
                              L_point.translation - center);
  // now do the global translation of the cells:
  KL_segment.translation = center;
  // the center of arc/disk will be in the center of disk
  KFL_arc.translation = LHKF.center;
  KFL_half_disk.translation = LHKF.center;
  // define the arc using radius and two points on it
  KFL_arc.shape = ARC( K_point.translation - center,
                      L_point.translation - center,
                      LHKF.scale);
  // now do the global translation of the cell:
  KFL_half_disk.shape = ARC( K_point.translation - center,
                              L_point.translation - center,
                              LHKF.scale);

  isIntersection = true;
} else {
  isIntersection = false;
}
}
}
// predicate used to find out if the instance is still valid:
PREDICATE {
  bool evaluate( REAL globalT, REAL localT, REAL dt)
  {
    return !isIntersection;
  }
}
} // K3

```

The dynamic IC for this model over its limited life span is defined as (see fig 84):

$$\mathbf{K}(t) = \begin{cases} K_1(t^{K_1}); P^{K_1}(t^{K_1}(t)) = true \\ K_2(t^{K_2}); P^{K_2}(t^{K_2}(t)) = true \\ K_3(t^{K_3}); P^{K_3}(t^{K_3}(t)) = true \end{cases}$$

The above example demonstrated a simple time-dependent hybrid model. This model incorporates cells expressed in different representations. The state of one of the cells changes over time. Changes of the structural state of the model are also reflected in the description of the model through a number of IC instances.

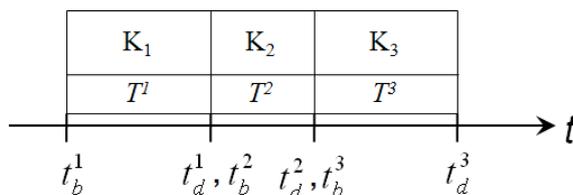


Figure 84: A dynamic IC over time (here the time parameter of the IC is its local time).

In this example we have presented a detailed description of all the IC instances depending on the mutual positions of the cells. In fact, we do not have to provide a description of a new IC instance for every structural state of the IC, if it is not considered to be relevant. Certain topological relations could become invalid over the course of the modelling process, but if these relations are not considered crucial for this particular model the user does not have to describe them with a distinct IC instance (see section 3.5).

5.2 An exemplar of a 2D dynamic IC model with dependencies

In this example (see fig. 85) we present a model similar to the one described in the previous section. The main difference here is the introduction of a simple dependency relation (a special case of a hierarchical dependency, introduced in section 3.5) between two cells (see fig. 86). For simplicity we may think of this model as consisting of two cells with their locations changing simultaneously.

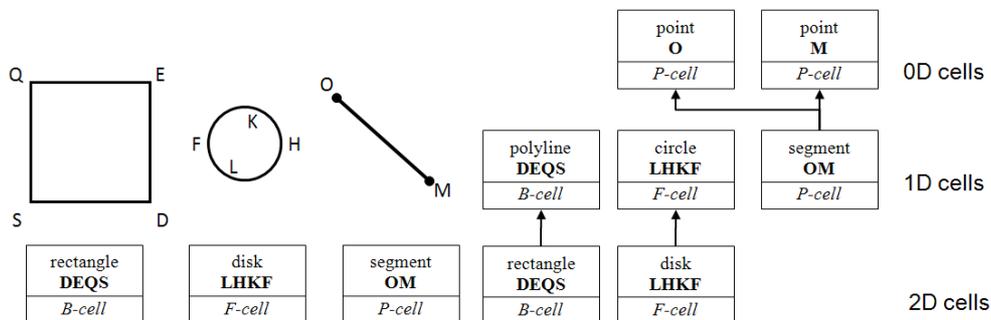


Figure 85: The cells and relations of the dynamic IC.

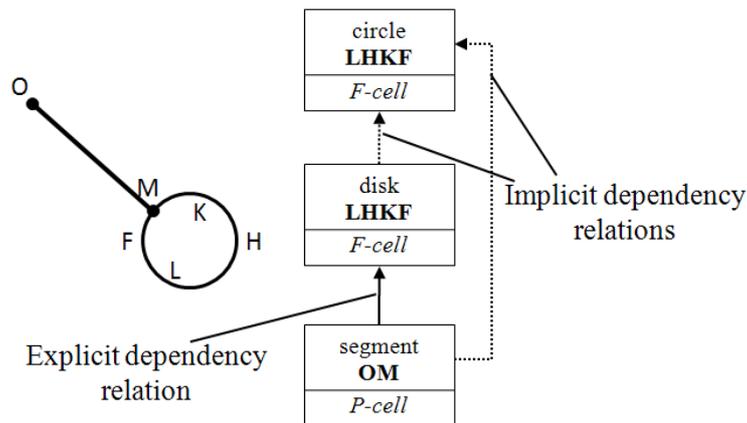


Figure 86: The cells and the dependency relations between them.

In this case we use an externally defined animation curve to modify the position of one of the cells instead of doing so with a procedurally defined motion. We start by describing some basic cells present in the model:

```
DEQS {
  type = B-CELL;
  // use one of the library objects setting up appropriate parameters
  shape = library::rectangle(...);
  dim = 2D;
  domain = {...};
  // we will only be using built-in transforms, thus no additional
  // parameters will be defined
  parameters {
    translation(...); // define initial translation
  }
} // DEQS

LHKF {
  type = F-CELL;
  // load the description of the FRep tree (XML, HyperFun etc)
  shape = frep::loadTree(...);
  dim = 2D;
  domain = {...};
  // custom parameters of the cell
  parameters {
    translation(...); // define the initial position
  }
  REACTIONS {
    // no reactions are defined for this cell
  }
} // LHKF

OM {
  type = P-CELL;
  shape = segment(...);
  dim = 1D;
  domain = {...};
  REACTIONS {
    // update the shape using the predefined animation
    update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
  }
}
```

```

    {
      // use the externally defined animation to retrieve
      // current position of the segment
      translation.relative =
        animations( "animCurve1" ).params(localT) );
    }
  } // OM

```

Next we define the IC template used for the subsequent description of the IC model:

```

// describe the initial instance, when there are not additional
// relations between the cells
INITIAL_TEMPLATE : TEMPLATE_IC {
  // references to a set of previously described cells
  CELLS {
    DEQS LHKF OM; // reference to independent cells
  }
  // a set of relations between the cells/parameters referenced in
  // the CELLS section
  RELATIONS {
    // the list of containment relations (will use implicit relations
    // from the definitions of DEQS, LHKF and OM)
    containment {
    }
    // the list of boundary relations (will use implicit relations
    // from the definitions of DEQS, LHKF and OM)
    boundary {
    }
    // list of dependency relations
    dependency {
      OM LHKF HIERARCHICAL LINE_DISK;
    }
  } // RELATIONS
} // INITIAL_TEMPLATE

```

The following state parameters are used to reflect the state of the model:

```

STATE_PARAMETERS {
  isLineRectIntersection : BOOL( false );
  isDiskRectIntersection : BOOL( false );
}

```

The first IC instance (fig. 87) is now defined as:

```

// use all the components of the template
K1 : INITIAL_TEMPLATE {
  // predicate used to find out if the instance is still valid:
  PREDICATE {
    bool evaluate( REAL globalT, REAL localT, REAL dt)
    {
      // try finding the intersection between the disk and
      // the rectangle using the variable of the IC
      isDiskRectIntersection = DEQS.domain.isIntersection(
        LHKF.shape.global);
      isLineRectIntersection = DEQS.domain.isIntersection(

```

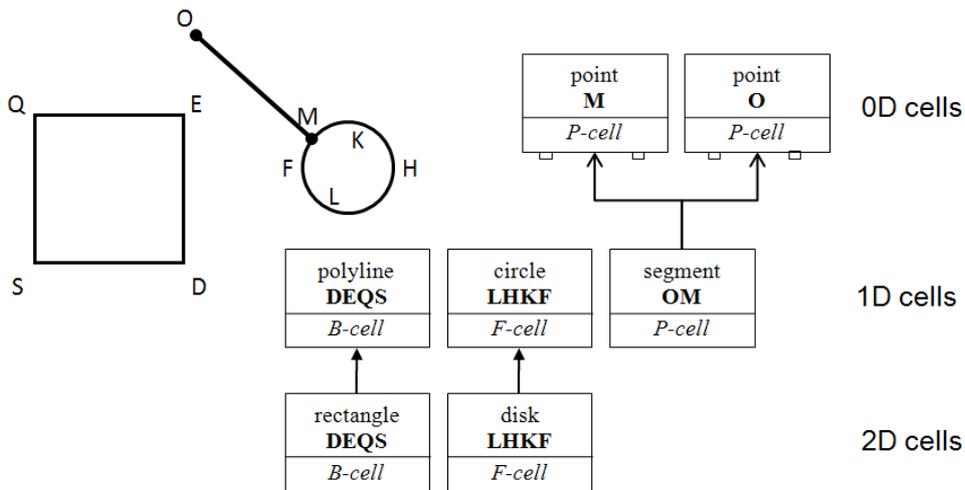


Figure 87: The first IC instance.

```

                                OM.shape.global);
    return !(isDiskRectIntersection || isLineRectIntersection);
}
}
} // K1

```

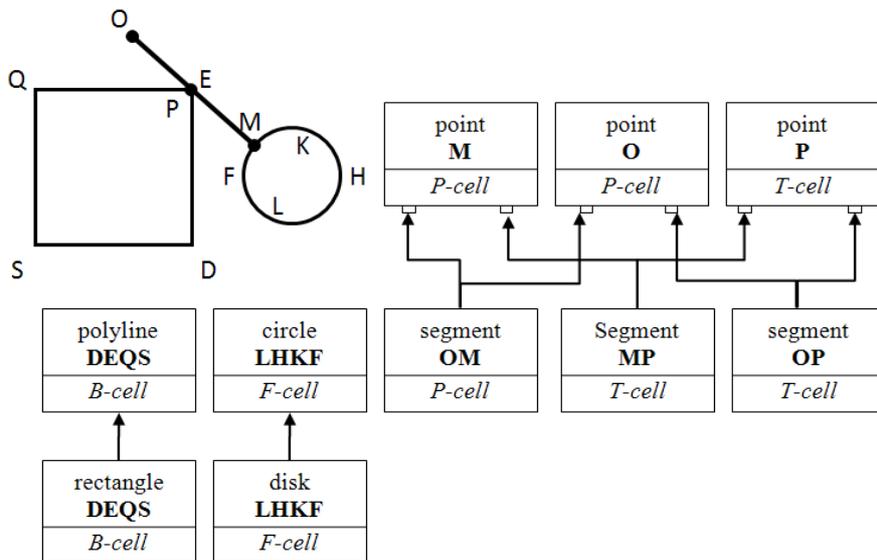


Figure 88: The second IC instance.

The second IC instance is activated when the segment OM has exactly one intersection point with the rectangle DEQS (see fig. 88). Thus, we introduce an additional T-cell:

```

P_point {
    type = T-CELL;
}

```

```

        dim = 0D;
        domain = {...};
        // the translation of this cell can be determined only after
the
        // the intersection test, this will be done in the instance
    }

```

Next comes the description of the second IC instance:

```

K2 : INITIAL_TEMPLATE {
    STATE_PARAMETERS {
        // true if only one point results from the intersection
        isOnePointIntersection : BOOL( false );
    }
    // references to a set of previously described cells
    CELLS {
        // add a new cell to the set of cells from IC template
        P_point;
    }
    // events processed by the instance:
    REACTIONS {
        // update the position of the T-cell (Post suffix as we want
        // it to be issued after all the cells have been updated)
        updatePost( REAL globalT, REAL localT, REAL lifeT, REAL dt)
        {
            isOnePointIntersection = false; // default value
            isDiskRectIntersection = true; // default value
            // try finding the intersection between the disk and
            // the rectangle
            INTERSECTION_INFO info1 = DEQS.domain.isIntersection(
                LHKF.shape.global);
            INTERSECTION_INFO info2 = DEQS.domain.intersect(
                OM.shape.global);
            // only one intersection point exists
            if (info2.intersectPoints.size == 1) {
info
                // set the position of the T-cell using the intersection
                P_point.translation = info2.intersectionPoint(0);
                isOnePointIntersection = true;
            }
            isLineRectIntersection = (info2.intersectPoints.size!=0);
            if (info1.intersectPoints.size == 0) {
                isDiskRectIntersection = false;
            }
        }
    }
    // the predicate used to find out if the instance is still valid:
    PREDICATE {
        bool evaluate( REAL globalT, REAL localT, REAL dt)
        {
            return !isDiskRectIntersection && isOnePointIntersection;
        }
    }
} // K2

```

The third instance has two additional intersection points, i.e., two addi-

tional T-cells (see fig. 89).

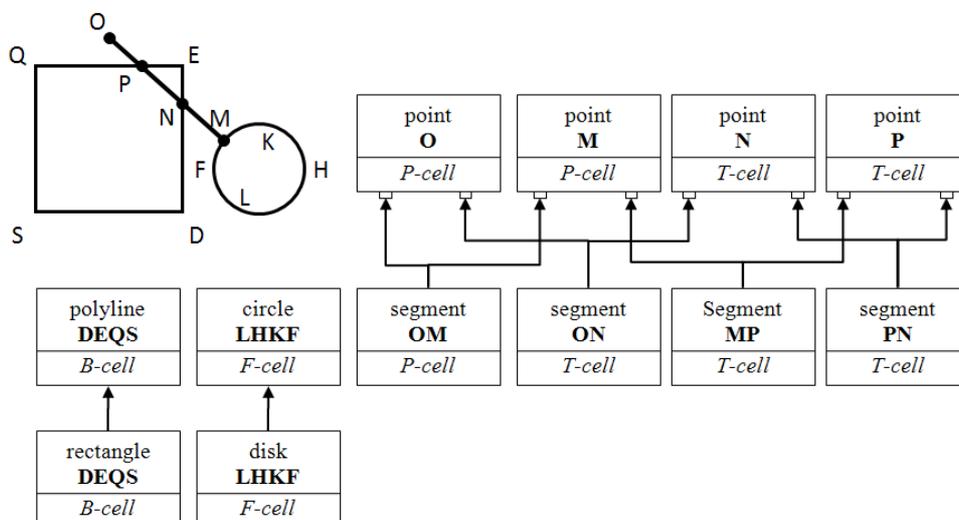


Figure 89: *The third IC instance.*

```

K3 : INITIAL_TEMPLATE {
  STATE_PARAMETERS {
    // true if two points result from the intersection
    isTwoPointIntersection : BOOL( false );
  }
  // references to a set of previously described cells
  CELLS {
    // copy all the cells from a template complex
    USE INITIAL_TEMPLATE.CELLS;
    P_point N_point;
  }
  // events processed by the instance:
  REACTIONS {
    // update the position of the T-cell (Post suffix as we want
    // it to be issued after all the cells have been updated)
    updatePost( REAL globalT, REAL localT, REAL lifeT, REAL dt)
    {
      isTwoPointIntersection = false; // default value
      isDiskRectIntersection = true; // default value
      // try finding the intersection between the disk and
      // the rectangle
      INTERSECTION_INFO info1 = DEQS.domain.isIntersection(
        LHKF.shape.global);
      INTERSECTION_INFO info2 = DEQS.domain.intersect(
        OM.shape.global);
      // only two intersection points exist
      if (info2.intersectPoints.size == 2) {
        // set the position of the T-cell using the
        // intersection info
        P_point.translation = info2.intersectionPoint(0);
        N_point.translation = info2.intersectionPoint(1);
        isTwoPointIntersection = true;
      }
      isLineRectIntersection = (info2.intersectPoints.size!=0);
      if (info1.intersectPoints.size == 0) {

```

```

        isDiskRectIntersection = false;
    }
}
}
// predicate used to find out if the instance is still valid:
PREDICATE {
    bool evaluate( REAL globalT, REAL localT, REAL dt)
    {
        return !isDiskRectIntersection && isTwoPointIntersection;
    }
}
} // K3

```

The fourth instance (see fig. 90) has a new T-cell. Hence, we add a new T-Cell:

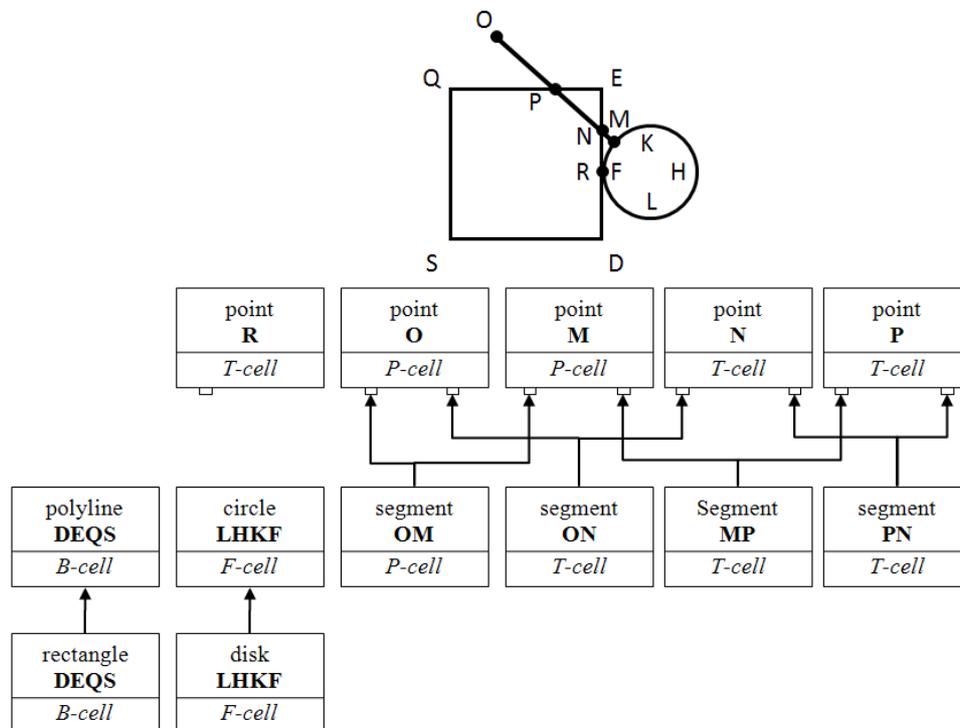


Figure 90: *The fourth instance.*

```

K4 : INITIAL_TEMPLATE {
    STATE_PARAMETERS {
        // true if two points result from the intersection
        isTwoPointIntersection : BOOL( false );
        // true if the disk only has one intersection point
        // with the line
        isOnePointDiskIntersection : BOOL( false );
    }
    // references to a set of previously described cells
    CELLS {
        // copy all the cells from a template complex
        USE INITIAL_TEMPLATE.CELLS;
    }
}

```

```

    P_point N_point R_point;
  }
  // events processed by the instance:
  REACTIONS {
    // update the position of the T-cell (Post suffix as we want
    // it to be issued after all the cells have been updated)
    updatePost( REAL globalT, REAL localT, REAL lifeT, REAL dt)
    {
      isTwoPointIntersection = false; // default value
      isDiskRectIntersection = true; // default value
      isOnePointDiskIntersection = false;
      // try finding the intersection between the disk and
      // the rectangle
      INTERSECTION_INFO info1 = DEQS.domain.isIntersection(
        LHKF.shape.global);
      INTERSECTION_INFO info2 = DEQS.domain.intersect(
        OM.shape.global);
      // only two intersection points exist
      if (info2.intersectPoints.size == 2) {
        // set position of the T-cell using intersection info.
        P_point.translation = info2.intersectionPoint(0);
        N_point.translation = info2.intersectionPoint(1);
        isTwoPointIntersection = true;
      }
      isLineRectIntersection = (info2.intersectPoints.size!=0);
      if (info1.intersectPoints.size == 1) {
        isOnePointDiskIntersection = true;
        R_point = info1.intersectionPoint(0);
      }
      isDiskRectIntersection = (info1.intersectPoints.size!=0);
    }
  }
  // predicate used to find out if the instance is still valid:
  PREDICATE {
    bool evaluate( REAL globalT, REAL localT, REAL dt)
    {
      return isOnePointDiskIntersection&&isTwoPointIntersection;
    }
  }
} // K4

```

The fifth instance (see fig. 91) is defined in a similar way. The resulting IC is defined as:

$$\mathbf{K}(t) = \begin{cases} K_1(t^{K_1}); P^{K_1}(t^{K_1}(t)) = true \\ K_2(t^{K_2}); P^{K_2}(t^{K_2}(t)) = true \\ K_3(t^{K_3}); P^{K_3}(t^{K_3}(t)) = true \\ K_4(t^{K_4}); P^{K_4}(t^{K_4}(t)) = true \\ K_5(t^{K_5}); P^{K_5}(t^{K_5}(t)) = true \end{cases}$$

Figure 92 shows the five instances of the IC and the sequence of transitions

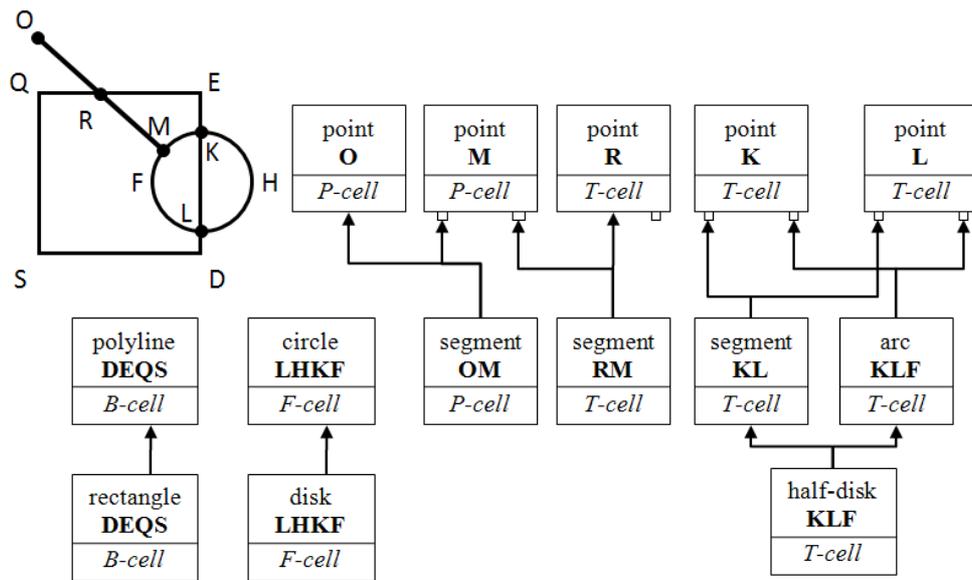


Figure 91: The cells and relations of the fifth IC instance.

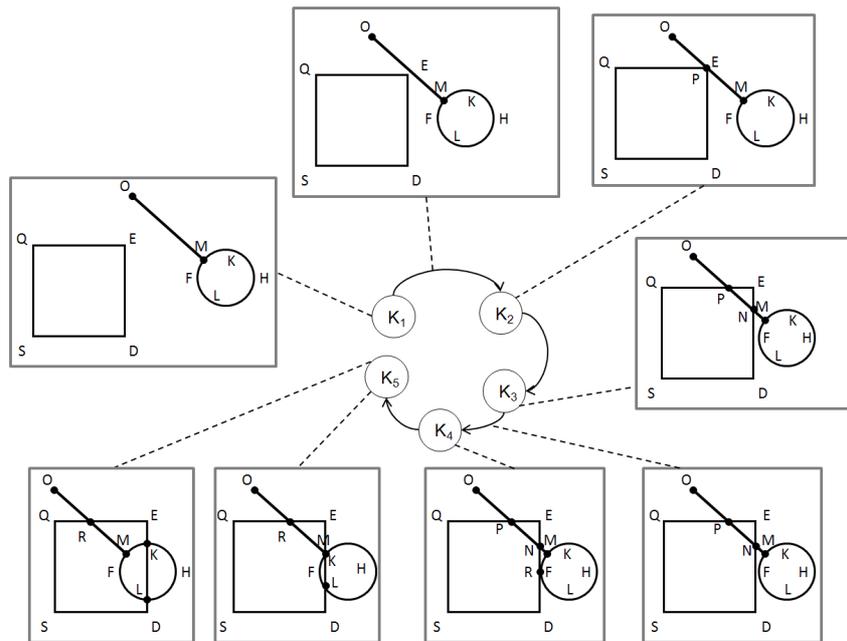


Figure 92: The IC instances and the transitions between them.

between them.

It is worth mentioning that not all the instances of the IC need to be defined. The freedom that the definition of a predicate provides, allows the users the flexibility of only having to define the conditions of interest to them. Even if the instance is invalid in a strict sense (e.g., if some new relations need to be

established), this does not mean the predicate of the instance has to indicate the invalid state of the instance being active. Users are given the opportunity to concentrate on the states of the model which they consider to be important.

5.3 Multidimensional dynamic models in the space-time domain

In this section we describe a mixed-dimensional model. In this model a set of lower-dimensional objects is used to produce a higher-dimensional object. This is made possible through the introduction of multiple modelling domains in section 3.4.1. The resulting higher dimensional object can be interpreted as a new shape or as a set of objects resulting from the metamorphosis between the original lower-dimensional objects. We take advantage of the existing space-time blending approach and we improve it in a number of ways.

5.3.1 Introduction to space-time blending

Space-time blending is based on the bounded blending operation (see section 2.2.3) performed in higher-dimensional space. The initial idea of space-time blending was introduced in (Pasko *et al.*, 2004b). Space-time blending allows us to perform transformations between shapes of different topology without necessarily establishing their alignment or correspondence. An example of space-time blending between a cross and two disks (see figure 93a) is shown in fig. 93b.

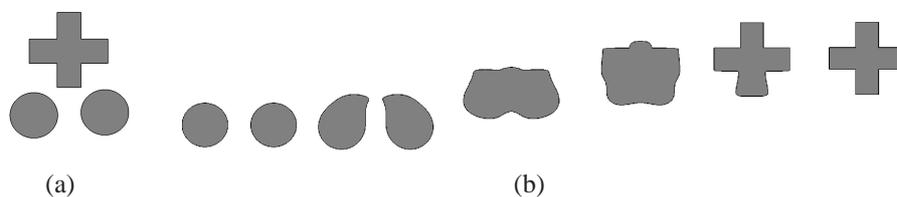


Figure 93: Space-time blending: (a) Two initial 2D objects (b) A set of intermediate objects generated using space-time blending.

Unlike a number of existing approaches (Sederberg and Greenwood, 1992; Sederberg *et al.*, 1993; Shapira and Rappoport, 1995; Cohen-Or *et al.*, 1996;

Zhang and Huang, 2000; Surazhsky *et al.*, 2001; Lazarus and Verroust, 1998) space-time blending is not based on any assumptions regarding the equivalence of the topology of the initial objects. It does not even require the shapes to be aligned or to have a vertex-to-vertex correspondence established by the modeller.

Let us illustrate the proposed space-time blending approach for the 2D shapes shown in figure 94. The two initial shapes are defined on the **XY** plane. Their extrusions are generated as 3D half cylinders. These 3D objects can be created in the space-time domain **XYT** or in a purely geometric domain **XYZ** (see section 3.4.1). In either cases the initial lower-dimensional shapes are interpreted as projections of higher-dimensional objects. These two objects are then used as operands to a bounded blending operation in a higher-dimensional space (see fig. 95a).

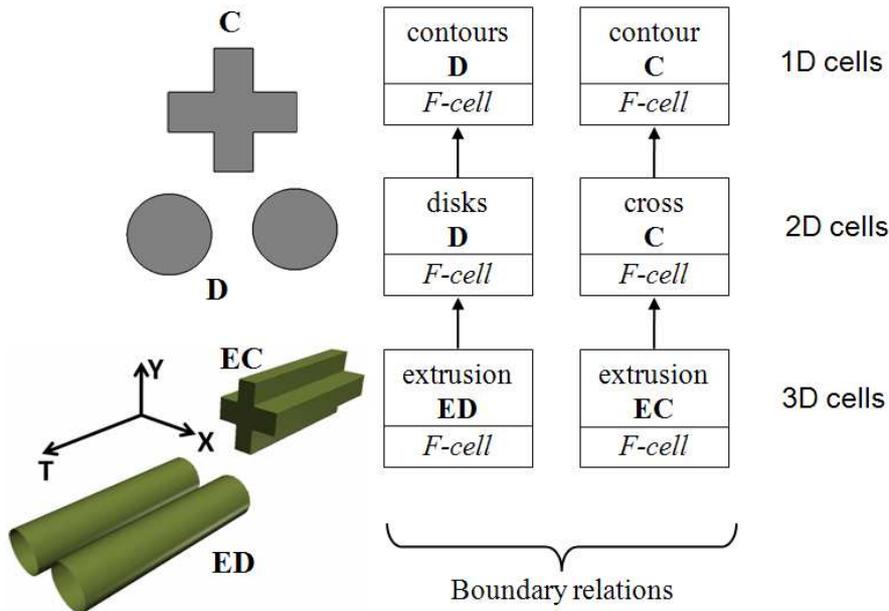


Figure 94: The IC cells initially present in the model.

5.3.2 The application of affine transformations in the space-time domain.

The original formulation of space-time blending in Pasko *et al.* (2004b) has several problems: a fast uncontrolled transition between shapes within the given time interval, the generation of disconnected components during the



Figure 95: The result of the application of a space-time blending operation: (a) Regular space-time blending (b) The proposed space-time blending with an additional affine transformation.

metamorphosis and the lack of intuitive user control over the transformation process Pasko *et al.* (2004c).

We have resolved some of these issues and improved the original technique in a number of ways (Pasko *et al.*, 2010). We have introduced an additional set of controllable affine transformations which are applied to the initial objects in space-time (see fig. 95b). This allows us to make a smoother transition from one shape to the other and to have better control over this transition. This is especially useful when the dimensions of the shapes vary significantly or when the distance between the initial shapes is large (see fig. 96)

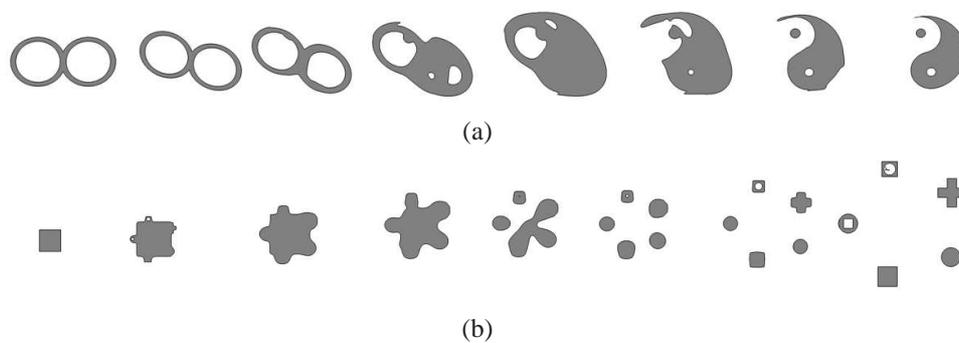


Figure 96: The cross-sections of the shape generated using the improved space-time blending: (a) a user guided rotation around the time axis to align object features (b) A user guided scale along the time axis.

The same improvements can be applied to 3D objects. In this case 3D slices of a higher-dimensional 4D object can be interpreted as intermediate shapes of the metamorphosis process between the initial 3D objects (see fig. 97). Without these transformations the volume of the intermediate shape needs to be significantly increased in order to avoid having disjointed compo-

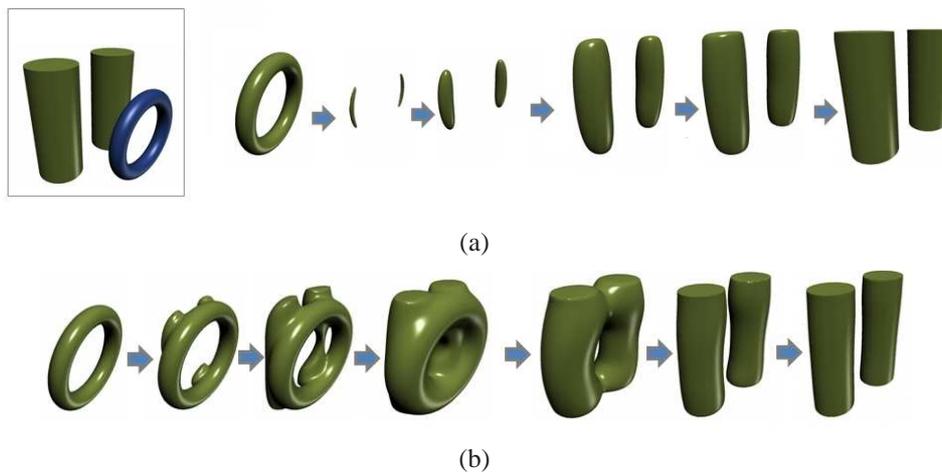


Figure 97: *The transition between 3D objects (a) Linear metamorphosis (Pasko et al., 1995). (b) Improved space-time blending.*

nents. But this increase of the volume leads to even faster transitions between the shapes. Thus, affine transformations provide more control over the interim modifications of the shape. Additionally, these transformations help us reduce the rate at which the transition between the shapes takes place. All required time-dependent affine transformations can be generated automatically based on the estimated bounding domains of the IC cells. More complex examples of possible transitions between 3D objects are depicted in fig. 98.

5.3.3 Additional time-dependent deformations.

Another inherent issue in the original space-time blending approach is the possible presence of disjointed components of the source and destination objects appearing during the transformation process (Fig. 99a). One way of resolving this issue is through the addition of user controlled deformations. The appearance of the disconnected component in fig. 99a can be explained by the significant difference in the distances between the initial torus and the final union of the two cylinders. The transition can be improved through the introduction of time-dependent deformations in addition to space-time blending. We can apply time-dependent deformations while transitioning from the source object to the destination object. For the example shown in fig. 99a, this can be done with the help of a non-linear space mapping (“warping”) in-

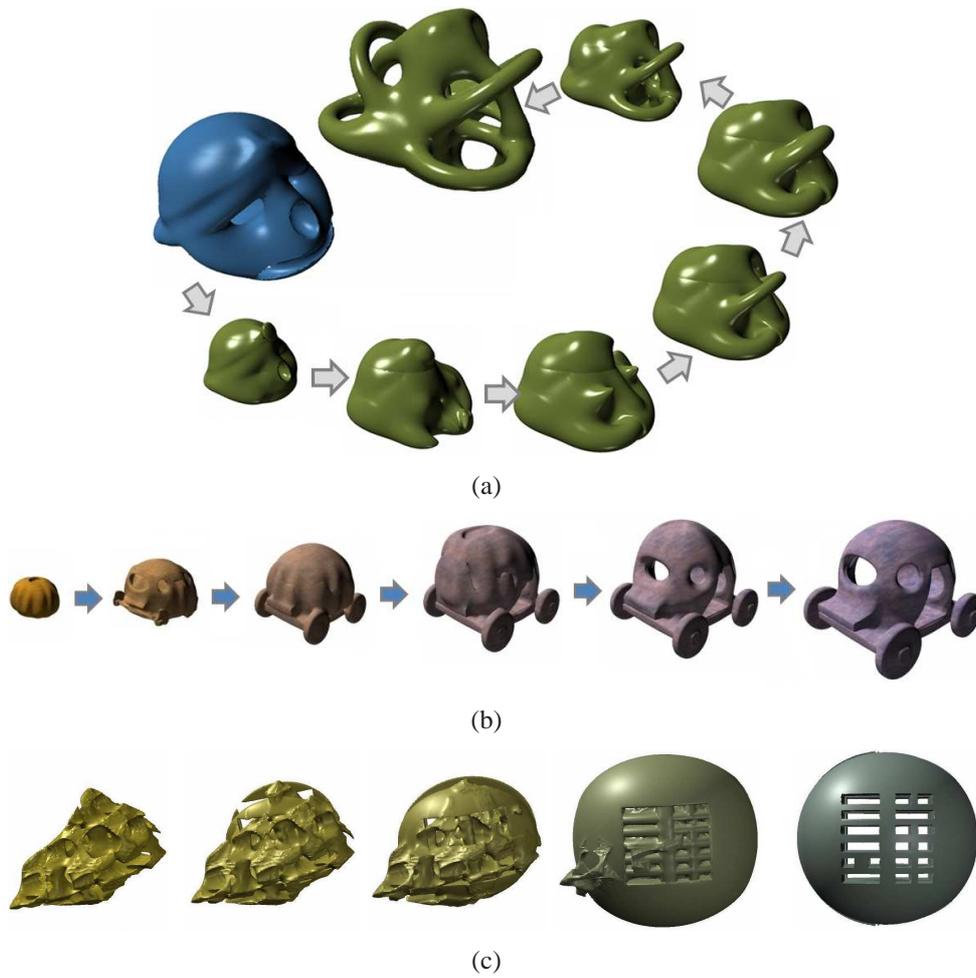


Figure 98: *Examples of transitions using space-time blending for 3D objects.*

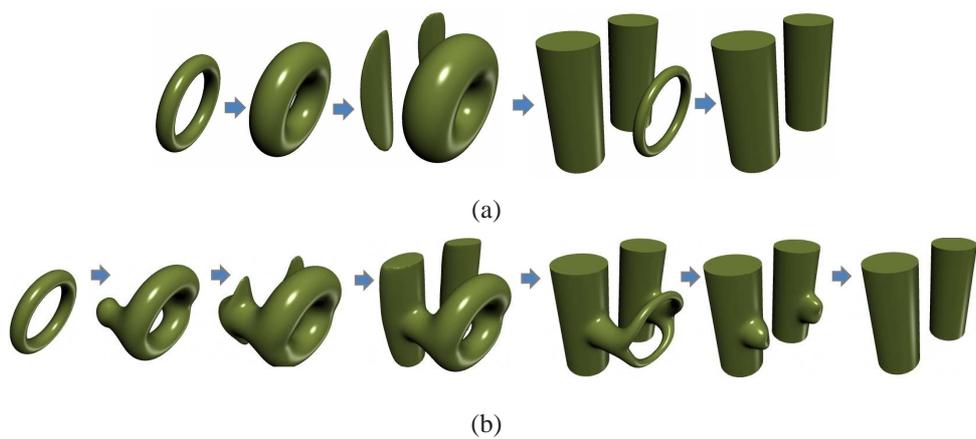


Figure 99: *Problems caused by disjointed components appearing during the transition: (a) Regular space-time blending (b) Space-time blending with additional deformations.*

tuitively controlled by two points (Schmitt *et al.*, 2003) as illustrated by fig. 99b. The number of these points is not restricted and can be chosen depending on the specific needs of the blending problem. In this case, additional deformations are added to modify the input shape of the space-time blending operation (see fig. 100). Although the user can define the control points for the deformation manually or interactively, these points can also be generated automatically based on the properties of the objects being blended. To do so we find a set of internal points with the extreme values of the defining function. These points are located inside the “thick features” of the model, i.e. the areas situated at the extreme distances from the object’s boundary:

$$\begin{aligned} \mathbf{D}_{src}^p &= \bigcup_{i=1}^{N_{src}} p_{srci} : F_{src}(p_{srci}) > 0, \\ F_{src}(p_{srci}) &> F_{src}(p_{srci} + \partial p); \|\partial p\| > 0 \\ \mathbf{D}_{dst}^p &= \bigcup_{j=1}^{N_{dst}} p_{dstj} : F_{dst}(p_{dstj}) > 0, \\ F_{dst}(p_{dstj}) &> F_{dst}(p_{dstj} + \partial p); \|\partial p\| > 0 \end{aligned} \quad (4)$$

where \mathbf{D} is a set of N points ($N_{src} = N_{dst}$) used to define the non-linear space-mapping, F_{src} and F_{dst} are the defining functions of the source and destination objects respectively. We find the locations of the aforementioned points performing a distance transform of the functional object using Euclidean metrics (Pasko *et al.*, 2010). The user may choose the number of points retrieved in this fashion and give a hint of how close to each other he/she wants the retrieved points to be (fig. 101). The retrieved points are located on the medial surface of the object.

5.3.4 Non-linear sampling in the space-time domain.

Due to the non-linear nature of the defining functions of the objects and the properties of the bounded blending operation, the transition between the objects can not be expected to be a linear process. But we can adjust the visual rate of this transition by performing non-uniform sampling over time. In the simplest case the time step can be adjusted depending on the estimated change of the area or volume of the shape. A modification of the time step is

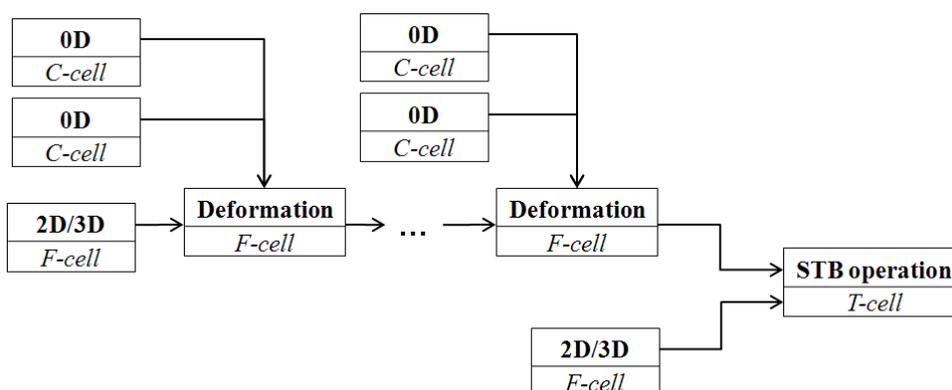


Figure 100: The dependency relations for an improved space-time blending with additional deformations.

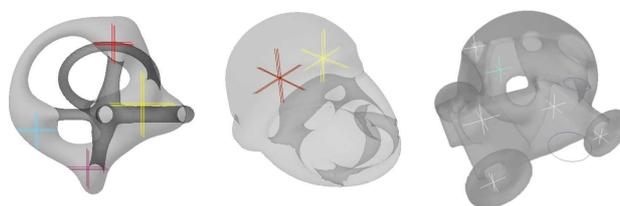


Figure 101: Examples of the extracted “thick features” (marked by crosses).

performed using a feature of the dynamic IC described in section 4.4.2. This mechanism allows us to request a modification of the global time using the events mechanism:

```

STB_CELL {
  // representation
  type = T-CELL;
  shape = IC::spaceTimeBlendCells(cellA, cellB, ...);
  // two 3D cells used as input, result is 4D
  dim ::= 4D;
  ...
  parameters {
    // the previous volume of the cross section
    shapeVolume : REAL(0.0);
    // the acceptable rate of volume change
    volumeRate : REAL (...);
    // how far back time needs to be rewind
    timeDivider : REAL(0.0)
  }
  // reactions to events
  REACTIONS {
    update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
    {
      // retrieve the cross section of the 4D space-time object
      SHAPE crossSection3D = makeCrossSection(shape, globalT);
      // estimate the volume of the 3D cross section
      REAL curVolume = getVolume(crossSection3D);
      // check if the volume has changed significantly
      if (shapeVolume > 0 &&

```


initiation of F-Cells was later integrated into the IC model and is described in section 5.7.

5.4 Implicit “Stand-ins”: a case of time-variant hybrid modelling

In this section we describe a hybrid model in which we demonstrate the interaction between an animated character represented by a polygonal mesh and a viscoelastic object represented by an FRep object. This experiment demonstrates the advantages of the IC framework which allows us to integrate models defined in different representations within one model and to establish dependency relations between these.

5.4.1 Introduction to the “stand-ins” technique

One of the advantages of hybrid models is the possibility to create animation effects which are quite hard to achieve using any single model representation. Here we mainly consider a problem of interaction between complex dynamic objects and viscoelastic substances. We model this interaction through the combination of animated polygonal meshes with FRep objects using our dynamic IC framework. An animated mesh is approximated by a convolution surface stand-in that is embedded within it or is attached to it. The motions of both objects are then synchronised using a rigging skeleton. We model the interaction between an animated mesh object and a viscoelastic substance, which is represented by an FRep object. This approach is aimed at achieving verisimilitude rather than physically based simulation. The adhesive behaviour of the viscous object is modelled using geometric blending operations on the corresponding FRep objects. Another application of this approach is the creation of metamorphosing FRep parts that are attached to an animated mesh. A further extension of this approach for the controlled metamorphosis of animated meshes is described in section 5.5.

Polygonal meshes and certain types of implicit surfaces can be animated using a rigging skeleton. We consider a skeleton as a platform for their in-

tegration into hybrid models. There are many candidates for such integration among implicit surfaces, namely soft objects, distance-based blobs, ellipsoids, convolution surfaces, constructive solids built from cylinders, spheres, and other primitives. The main requirements for an implicit surface are: a relatively simple defining function, which is fast to evaluate, easy to manipulate using skeletons and an absence of bulges and other unwanted artefacts, which require additional processing. All these requirements are satisfied by convolution surfaces (Bloomenthal and Shoemake, 1991; McCormack and Sherstyuk, 1998) which we choose for our purposes²¹. We propose to embed an implicit convolution surface inside an animated mesh or to attach it to the mesh such that the motions of both types of object are synchronised. The objects can either share a common skeleton or have individual synchronously moving skeletons.

An embedded convolution surface has to closely approximate the embedding mesh such that its motion requires no changes or minimal changes of the convolution surface parameters. This may require a procedure for fitting a convolution surface to an initial mesh taking into account its specified motion, which can be achieved using a global minimisation of the overall algebraic distance of the mesh nodes from the convolution surface. The interaction of a viscous object with an animated object is modelled using geometric blending operations on the corresponding implicit surfaces. Note that the initial animated mesh is rendered in the final animation together with the blending surface, which creates the visual effect of the blending of the mesh itself. Thus the embedded convolution surface serves as an implicit stand-in for the animated mesh. Unlike most other approaches to the resolution of this modelling problem, our approach is not aiming at physical correctness. In the areas of computer animation and digital special effects production it is a well established fact that physically correct simulation is often an inappropriate technique to use, as it often interferes with the intended development of the narrative. Animators are often looking for some form of believable semblance of reality (i.e., verisimilitude), which is inspired by physical reality but bends this reality to allow them to advance the story narrative. Instead of a physically correct simulation what is required is a set of techniques and tools that

²¹However other types of implicit surfaces could also be used by the proposed approach.

would allow the animator to alter and to fine-tune reality. These may be physically inspired, but not physically correct, and must be able to be directable by the animator, so that they produce the desired visual effects. Physically-correct simulation techniques can often be combined with physically-inspired verisimilitude techniques but they must be directable by the artist and subordinated to the story-telling process. Additionally when such techniques are used at the development stage of computer animation and digital effects sequences or in a computer game they must produce believable visual results in real-time or near-real-time. We aim to provide the animator with a simple tool based on purely geometric methods which allows the creation of complex animations satisfying the specified requirements.

5.4.2 Background

Historically a number of authors have used implicit surfaces for character animation. Elliptical blobs for skeletal animation were used by Jim Blinn (Blinn, 1982) back in 1982, where the transformation of the blob is inherited from the transformation of the joints of the skeleton. Opalach and Maddock used blobby objects for the easy definition of animated characters (Opalach and Maddock, 1995). However, their method is ill-suited for controlling the resulting “blobby” mesh. In addition, a large number of primitives are usually needed to model an appropriate mesh.

One of the earliest attempts of using hybrid modelling involved embedding mesh objects into implicit surface primitives (Singh and Parent, 1995) to implement polyhedral object deformations of articulated deformable bodies. Skeleton-based implicits for non-polygonal animated objects were examined in (Cani-Gascuel, 1998), where skeletal geometric primitives that produce distance fields were used for character animation - although this technique may lead to C^1 discontinuities in the resulting surfaces. The coating of arbitrary animated models by implicit surfaces, employed in this technique, is not always acceptable to animators. We consider our approach complementary to the coating technique. Mixing of implicit surfaces and polygonal models was performed in (Leclercq *et al.*, 2001). In this work specific regions of an animated mesh were deformed using implicit primitives attached to the animated

skeleton. Polygonal meshes and implicit primitives were also combined together in a HybridTree (Allègre *et al.*, 2006) using blending, Boolean and other operations supported by the conversion procedures between two different models. However, embedding, attachments and skeleton-based motion synchronisation of meshes and implicits as well as their implementation in a general-purpose animation system were not directly addressed.

Implicit surfaces were also used for the approximation of polygonal meshes using different approaches, such as Radial Basis Functions (RBFs) (Savchenko *et al.*, 1995) and Multi-level Partition of Unity implicits (MPUs) (Ohtake *et al.*, 2003). These methods generally work well with static meshes, but are less suitable for animation because dynamic models require per-frame re-fitting and can not be easily edited by the user due to the complicated handling of the implicit surface.

One of the interesting alternatives among implicit surfaces is that of convolution surfaces (Bloomenthal and Shoemake, 1991). Convolution surfaces can be smoothly blended with each other and provide a good approximation for polygonal meshes typical of skeletal characters with axial symmetry (McCormack and Sherstyuk, 1998). Our hybrid modelling approach takes advantage of convolution surfaces with line segment skeletons.

Traditionally physical simulation techniques were used for the modelling of interactions between dynamic entities and viscous objects or for controllable manipulation of viscous objects alone. A number of authors have proposed solutions to this problem (Foster and Fedkiw, 2001; Clavet *et al.*, 2005; Mcnamara *et al.*, 2004; Carlson *et al.*, 2004; Shi and Yu, 2005; Thürey *et al.*, 2006) - to name but a few. Fluid simulation using the Lagrangian or Eulerian approach allows for the creation of realistic animations, but usually this requires a lengthy simulation process and often provides the user with poor artistic control (i.e. poor directability) of the resulting effects.

It is often the case that a combination of different techniques is used for the emulation of viscous materials. In our approach we simulate objects composed of viscous materials using a geometric blending between the implicit objects generated from given polygonal meshes. We aim to provide the user with a simple tool which allows the creation of complex animations with con-

vincing visual results in real-time or near-real-time.

5.4.3 Problem statement and approach outline

Our approach relies upon dynamic hybrid modelling combining BRep polygonal meshes with FRep objects. In general, there are three main ways of achieving this:

1. By coating of BRep meshes with FRep objects.
2. By embedding FRep objects inside BRep objects.
3. By attaching external FRep objects to BRep objects.

As was mentioned above coating was discussed in (Cani-Gascuel, 1998). We apply embedding in order to achieve blending effects and attaching is used in order to construct the metamorphosing parts of hybrid models. Additionally, there is an important constraint that applies to our approach, namely the near real-time rendering of all hybrid models.

Let an animated object be defined by a polygonal mesh (see fig. 102a), with a rigging skeleton (see fig. 102b), skinning information (see fig. 102c) and a set of animation transformations for its skeletal nodes. A rigging skeleton is a set of hierarchically connected joints used to specify the motion of a mesh model in an animation sequence. If there is no skeleton provided, it can be automatically extracted from the polygonal mesh using one of the published techniques (Katz and Tal, 2003; Liu *et al.*, 2003; Baran and Popović, 2007).

An important application area for embedded implicit surfaces is the modelling of viscoelastic object adhesive behaviour in its interaction with an animated mesh object. To obtain visually plausible results with near real-time preview, the mesh object is replaced with an implicit stand-in. Geometric blending is then applied between the FRep entities representing both interacting objects.

A viscoelastic object can be represented either by an FRep objects or by another polygonal mesh (which has to be converted to an FRep object). We

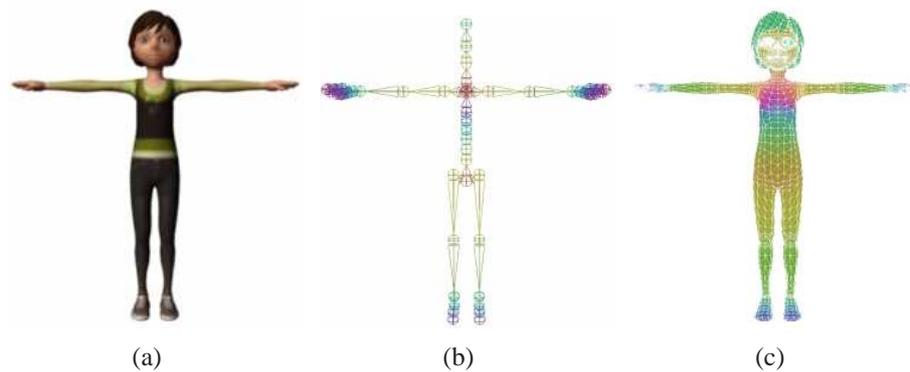


Figure 102: *Animated mesh information: (a) Polygonal mesh, (b) Rigging skeleton, (c) Skinning information. Model "Andy" courtesy of John Doublestein.*

will mainly concentrate on the former case to simulate such viscous substances as jam, honey or tar, and to show how such liquids interact with an animated object. Thus we will deal with the adhesion of the liquid to the surface, its stretching following the object's motion and other related topics.

Natural controllable blending is one of the best-known and useful properties of implicit surfaces (see section 2.2.3). We will use this property for modelling the adhesive behaviour of the liquid substance. This, in general, assumes the conversion of the animated mesh into an FRep object. However, an exact conversion of this type is a complex task. Instead we take advantage of our hybrid model, which includes a polygonal mesh and an approximation of this mesh by an FRep object embedded within it using a fitting procedure. It is impractical to perform this fitting to the mesh for each frame of the animation. Thus, it is preferable that an FRep objects is made to follow the motion of the animated mesh. A convolution surface satisfies this requirement when its skeleton is built using the rigging skeleton of the animated mesh and the motions of both skeletons are synchronised. This derived convolution surface can be blended with the FRep object, representing the viscous liquid, to mimic its adhesive interaction with other objects. The fitting procedure provides the convolution surface with a minimal distance measure to the mesh. This is required in order to create the visual effect of the mesh being blended with the viscous liquid.

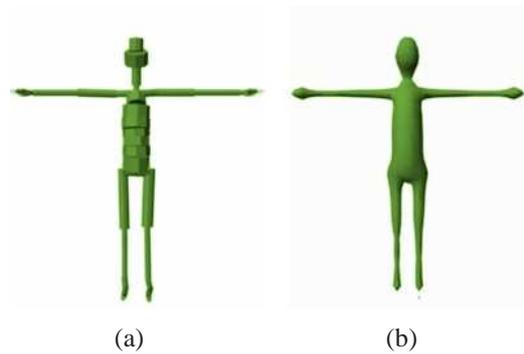


Figure 103: *Initial approximation: (a) The initial placement of bounding volumes inside the mesh, (b) The shape produced by convolution surface.*

A detailed description of convolution surfaces is provided in (Bloomenthal and Shoemake, 1991; McCormack and Sherstyuk, 1998). The main advantage of a convolution surface is the smooth transition between its parts that are defined by different skeletal elements (fig. 103b). When moving skeletal elements, the convolution surface follows their movement quite naturally, which is useful in animation (see fig. 104). We call this FRep model that is used for the approximation of the mesh an “implicit stand-in”.

The animated stand-in is then used in the model together with an object representing the liquid substance. Then a blending union operation²² is applied to these two objects. In order to achieve the desired effect.

5.4.4 The proposed approach

The proposed solution, outlined in section 5.4.3, can be subdivided into the following steps:

1. The creation of the initial approximation of the given mesh with bounding volumes using the skeletal information.
2. The tuning of the initial approximation.
3. The creation of an embedded convolution surface for the initial polygonal mesh.
4. One of two application steps: (a) the definition of the blending between

²²The blending union was described in section 2.2.3.

the convolution surface and the viscous object for the modelling of the adhesive behaviour of this viscoelastic object and its interaction with an animated object or (b) the creation of metamorphosing implicit parts for an animated mesh.

Each step requires the rendering of the current convolution surface and either the blending surface or the attached convolution surface. Note that both application steps can be performed together, when an animated mesh with attached implicit parts interacts with a viscous material.

The initial mesh approximation

As was mentioned earlier, the procedure for embedding an implicit surface inside the mesh requires a global minimisation of the algebraic distance measure between the mesh nodes and the convolution surface. As the first step of the global minimisation procedure, we can estimate the parameters of the convolution surface using the available information. For the initial approximation we use the rigging skeleton. Given the set of bones of the rigging skeleton, where each bone is a line segment in 3D space, we use the set of these segments as the basis for an initial convolution skeleton. We denote the start and end vertices for each such a skeletal segment as markers. To calculate the radius of the convolution surface for each segment, we calculate the minimal distance between each line segment specified by the markers and the polygonal mesh. At this stage we can build bounding volumes around each line segment for the real-time preview of the convolution surface. Each bounding volume is fitted inside the mesh in its initial position. Rendering these bounding volumes helps the user to better understand how the resulting approximating convolution surface is embedded into the mesh (see fig. 103).

In the next step we perform a global optimisation to achieve a better approximation of the given polygonal mesh using the embedded convolution surface. In order to achieve this we solve the constrained least-squares problem. We apply a numerical search in the n -dimensional space of the convolution parameters. We use the constrained Levenberg-Marquardt method (Kanzow *et al.*, 2005) to solve this problem. Usually the search procedure needs to be performed only once for the character's bind pose.



Figure 104: *The synchronised motions of the embedded convolution surfaces during the animation process.*

Further details regarding the approximation procedure and possible improvements of the approximation are described in detail in (Kravtsov *et al.*, 2010a)

The creation of a convolution surface

The embedded convolution surface is created by using the segments of the skeleton retrieved during the approximation procedure. For rendering purposes we use a polygonisation procedure, which provides an approximation of the implicit surface as a polygonal mesh. For relatively simple skeletons the polygonisation of the convolution surface can be obtained in near real-time. As the segments of the convolution skeleton are transformed relative to the transformation of the rigging skeleton, the motion of the convolution surface is synchronised with the motion of the animated mesh (see fig. 104).

We automatically perform the approximate convolution surface fitting only for the bind pose on the first frame of the animation. Thus, during the animation process the bounding volumes and the convolution surface itself may not fit inside the mesh. This could happen because the distances between the mesh vertices and the bones change noticeably for those vertices that are influenced significantly by more than one joints. Such vertices are usually positioned near the skeleton joints. Performing fitting of the convolution parameters for each key-frame of the animation can be a time-consuming process. This also means that each time the user adds a key-frame to the animation sequence the fitting procedure has to be repeated for these new frames. Thus, we let the user choose the key-frames for which refitting needs to take place - for instance, when the distance between the convolution surface and the bone exceeds the distance between the bone and the mesh. The re-estimated pa-

parameters are updated at the key-frames for the convolution primitives and then they are interpolated during the playback of the animation sequence. This allows the user to concentrate on the process of mesh animation by decreasing the delays caused by the implicit surface re-fitting. Also, there is an opportunity for the user to assign custom values to the parameters of the implicit surface over time - for instance, to change the parameter controlling the overall surface radius. This can be used to achieve a desired artistic effect for a particular animation sequence.

Applying the blending operation

As the first application of our technique, we simulate the interaction of a viscous object with an animated object using the blending union of two implicit surfaces. As we mentioned above, the implicit surface corresponding to the initial mesh is an embedded convolution surface. The second implicit surface representing the viscous object can be modelled using a set of implicit primitives. If both defining functions have distance properties, the shape of the surface resulting from the blending operation depends on the distance between the original implicit surfaces. The further the objects are from each other the less they deform. There exist three main phases of object interaction: the “continuous interaction” phase when the two implicit surfaces form a single blend shape (see fig. 105a), the “separation of two objects” phase (see Fig. 105b) and the “objects’ reciprocal attraction” phase resulting in the directional deformation which decreases proportionately to the distance between the two objects (see Fig. 105c).

A blending union can dramatically change the resulting surface and its topology. As a result of the mutual deformation, a part of the convolution surface embedded within the mesh becomes visible thus contributing to the material interacting with the mesh. Thus, the quality of the initial approximation of the mesh by the convolution surface does not play a significant part in this application. It is much more important to fully embed the convolution surface into the mesh when no deformation is applied.

Modification of the blending parameters produces an effect visually mimicking the viscous object’s physical parameter adjustment (see fig. 106). Thus, the user can control a specific phenomenon by modifying a meaningful

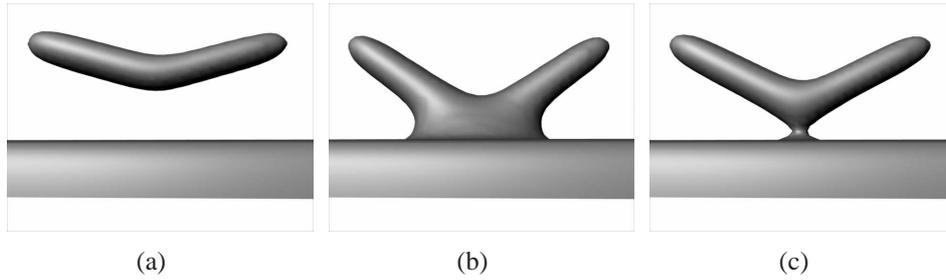


Figure 105: *Phases of interaction between animated objects without blending (left) and with blending (right): (a) Two implicit surfaces and a single blend shape during blending, (b) The boundary case before the two shapes separate, (c) Two separate shapes with some deformation showing the objects' reciprocal attraction.*

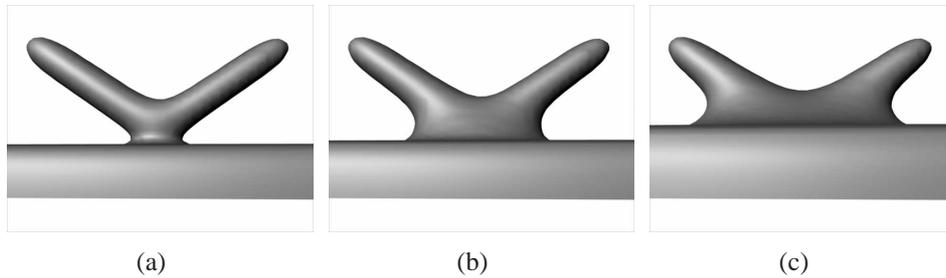


Figure 106: *Viscosity: (a) low, (b) medium, (c) high.*

set of parameters. Then, instead of using the abstract parameters of geometric blending (see section. 2.2.3), the user can operate with intuitive parameters representing the liquid viscosity or the gravitational force. A set of predefined templates for different materials (such as tar, honey, oil, etc.) allows the user to achieve easier control over the interaction process.

The combination of the aforementioned steps allows us to achieve the desired result. Figure 107 illustrates the process of hybrid modelling. A more detailed description of this technique is provided in (Kravtsov *et al.*, 2010a,b).

In the following section we provide a description of this model using our dynamic IC framework.

5.4.5 The IC model definition

Finally, we can present a description of the dynamic IC model using our approach (see section 4.3). Different states of the model are shown in fig. 108.

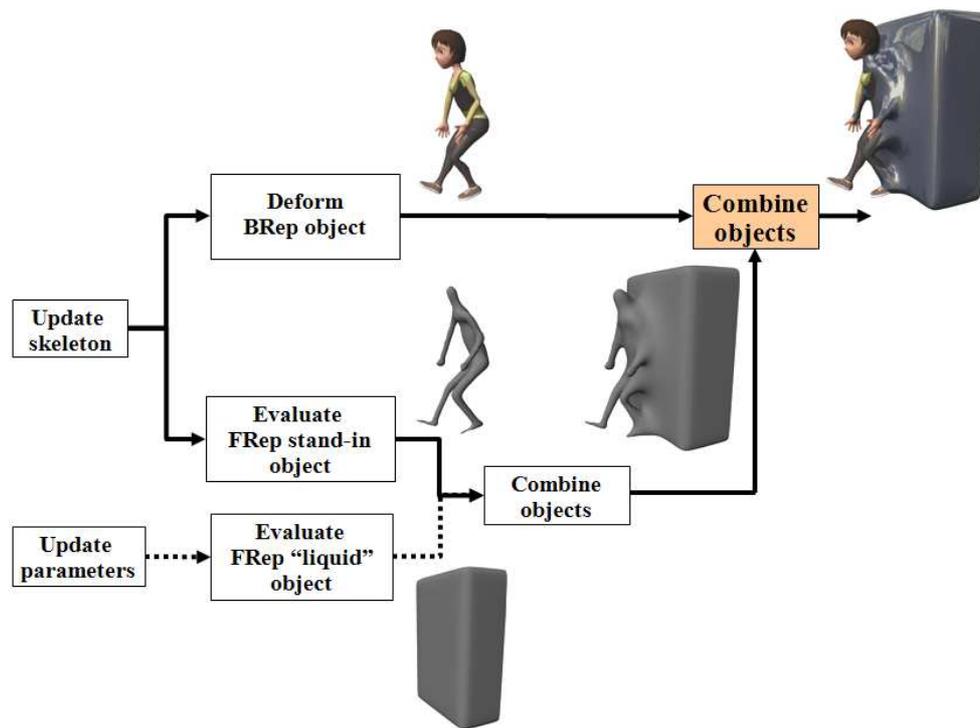


Figure 107: *Proposed approach outline.*

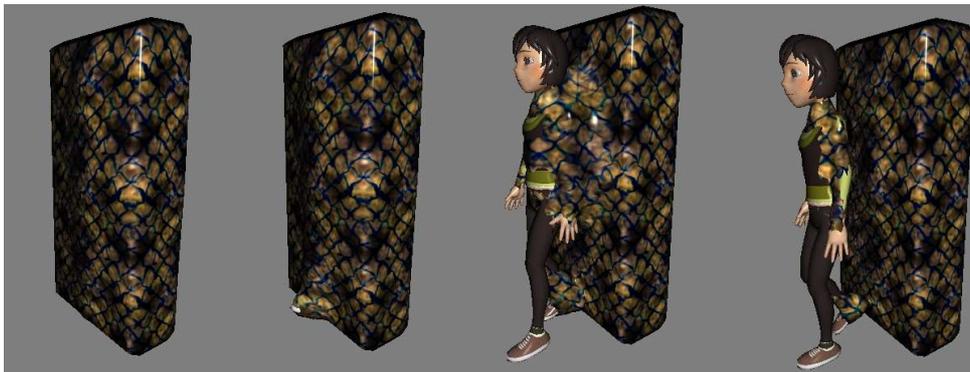


Figure 108: *The different states of the model.*

The IC model consists of (see figures 109 and 110):

- A deformable “box” containing a character at the beginning of the scene.
- A character called “Andy” (represented as a mesh) and her embedded stand-in convolution surface (Kravtsov *et al.*, 2010a).
- A cell called “Liquid on Andy”, which is used to mimic the viscoelastic behaviour of the “box” object interacting with “Andy”.

Both “Andy” and her implicit stand-in are controlled by the same skeleton through two dependency relations. Thus, motions of both objects are automatically synchronised using the common skeleton. It is worth noting that the shapes of both synchronised objects are represented by BRep and FRep cells. This arrangement would not be possible in the majority of existing animation systems.

The dynamic IC has two instances:

- “Andy” is placed inside the box and starts walking out of it. While she is walking out a liquid-like substance remains attached to her as though it was adhering to her.
- When “Andy” is completely outside the box (i.e. when there are no intersections between the animated mesh and the box) the liquid is no longer adhering to her.

The topological relations for the first instance of the IC are presented in figure 109. The dependency relations for this instance of the IC are outlined in figure 110. “Implicitly established” dependencies appear because “Liquid_on_Andy” is the result of blend operation between the existing cells.

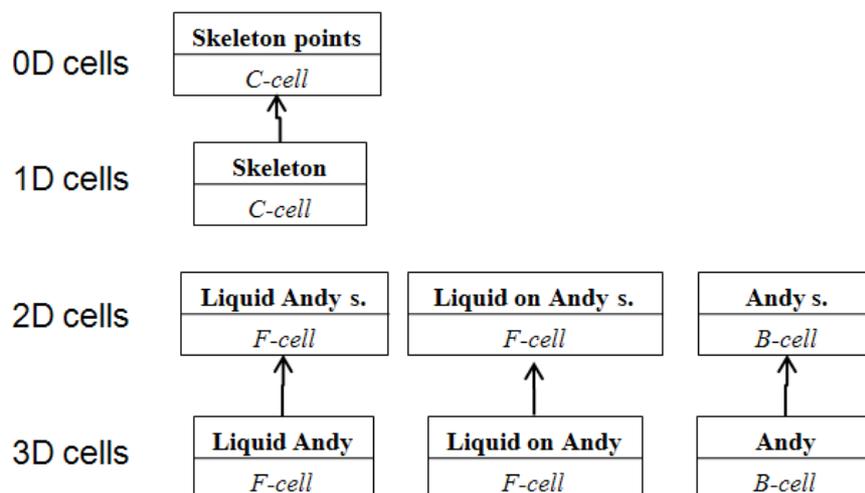


Figure 109: *The topological relations of the model.*

It is apparent from the illustrations that both “Andy” and “Liquid Andy” are dependent on the “Skeleton” cell. The skeleton controls the deformation of the cell “Andy” as well as the shape of the cell “Liquid Andy” (this cell is

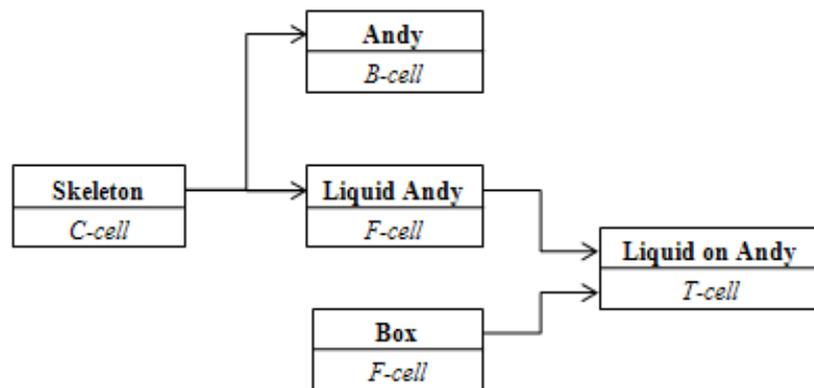


Figure 110: *The dependency relations of the model.*

the implicit stand-in of “Andy”). Thus the motions of both aforementioned dependent cells are synchronised over time. A blending union between the “Liquid Andy” and “Box” cells results in a liquid-like shape partly covering the animated “Andy” cell. This allows us to produce an interesting effect mimicking the viscoelastic behaviour of the “Box” object.

We use time-spans in order to define the interval of time when the walking animation is played:

```

// name of the model (can be used in other models):
name = ANDY_WALK_MIRROR
TIMESPANS {
  // the time-span used for the walking out animation (initialised
  later)
  walkTimespan : TIMESPAN;
  // the time-span used for the animation of the blending parameters
  // (initialized later):
  blendTimespan : TIMESPAN;
}

```

This time-span starts at the moment of the first IC instance activation:

```

// events processed by the instance
REACTIONS {
  //
  init( REAL globalT)
  {
    // walk span starts in this instance
    walkTimespan.start();
  }
  ...
}

```

Times-pans allow us to start certain animation sequences or countdowns at specific moments of time and to use their local time within them²³.

²³Alternatively we might also use animation sequences that do not depend explicitly on

The second IC instance is similar to the first, only this time the blending parameters between the “liquid Andy” (her implicit stand-in) and the liquid box are decreased relative to the distance between the skeleton and the liquid box:

```
// events processed by the instance
REACTIONS {
  // update the distance
  update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
  {
    // update the distance to the box
    distanceToBox = Box.domain.distance(Skeleton.curDomain);
  }
}
```

A detailed description of this IC model is provided in Appendix B.



Figure 111: A set of examples of the dynamic hybrid modelling technique.

The demonstrated approach, relying on implicit stand-ins using hybrid representations and dependencies between various parts of the model, can be applied in order to solve a number of problems (Kravtsov *et al.*, 2010a,b). A selected set of examples is provided in figure 111.

5.5 The controlled metamorphosis of animated meshes

In this section we introduce an approach which allows us to produce with great ease metamorphosing transitions between animated meshes of arbitrary topologies using hybrid models. Here we use the meshes of the objects as well as their skeleton animations. As a result we are able to generate metamorphosis animations of time-varying meshes with arbitrary topologies in near real-time. The approach presented here relies on the “stand-ins” technique described in section 5.4.

time, but are started just like time spans/timers (i.e. animation events).

5.5.1 Introduction

As we outlined in section 5.4, polygonal models animated using an underlying skeleton are widely used in computer animation. This approach, combining a set of simple skeletal deformations, allows the artist to produce complex animation sequences in a relatively easy way. However, performing complex transitions between arbitrary animated meshes remains a challenging problem. Existing shape blending techniques allow artists to perform limited transitions between a set of so called “blend-shapes”, but this is a rather limited approach, as the topologies of these blend-shapes need to be matched precisely. There is a set of established techniques to perform metamorphosis (3D morphing) between static 3D meshes Lazarus and Verroust (1998). Some of the existing methods overcome the significant limitations of the shape blending approach, but most of them cannot be easily applied to animated meshes. Our approach takes advantage of hybrid models, allowing us to produce with great ease metamorphosing transitions between animated meshes with arbitrary topologies.

5.5.2 Method Outline

As was mentioned earlier BRep meshes can be easily animated by an artist but such effects as metamorphosis cannot easily be performed. On the other hand, we know that metamorphosis can be easily performed between FRep entities (see section 5.3), which are harder to animate using the tools available to artists. The key idea of our method is based on the “stand-ins” technique described in section 5.4. We take advantage of both BReps and FReps, switching between these representations contained within a single hybrid model depending on our needs. In order to achieve this, we approximate the animated meshes using “implicit stand-ins”. This is done using a single pose of each mesh. The actual metamorphosis is then performed between the FRep entities approximating the animated BRep meshes. Thus, for the metamorphosis between the source and destination animated meshes we perform:

1. A smooth transition from the animated source mesh of a B-Cell to its functional approximation by an F-Cell (see K_2 in fig. 112);

2. A continuous transition from the functional approximation of the source mesh (F-Cell) to the functional approximation of the destination mesh (F-Cell), shown in the middle (K_3) of figure 112;
3. A transition from the functional approximation of the destination mesh (F-Cell) to the animated destination mesh (B-Cell). This final transition (K_4) is shown at the bottom of figure 112.

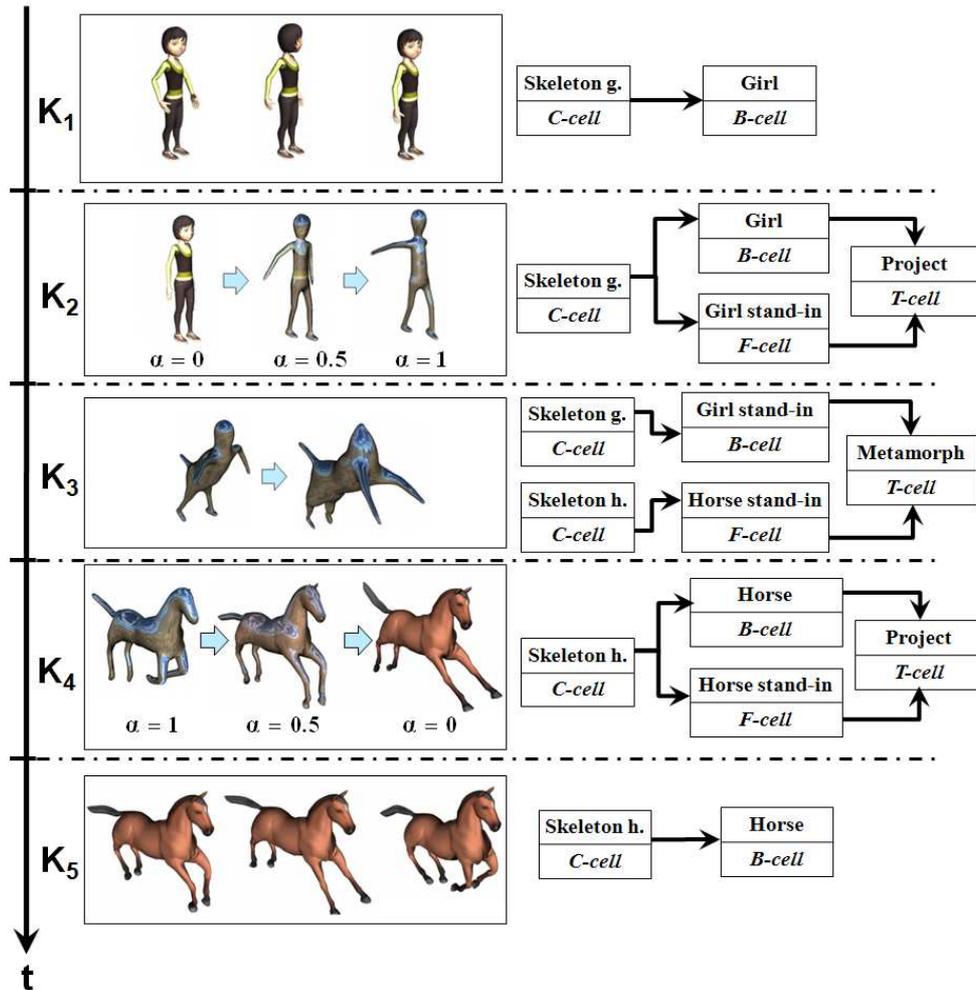


Figure 112: All IC instances of the controlled metamorphosis example together with the dependency relations.

In order to produce a smooth transition from the mesh contained in a B-Cell to an FRep entity contained in an F-Cell (step 1), we project the vertices of the mesh onto the approximating “stand-in”. We use the per-vertex skinning and normal information to retrieve an appropriate position for every vertex of the mesh on the surface of the “stand-in”. This can be done using a

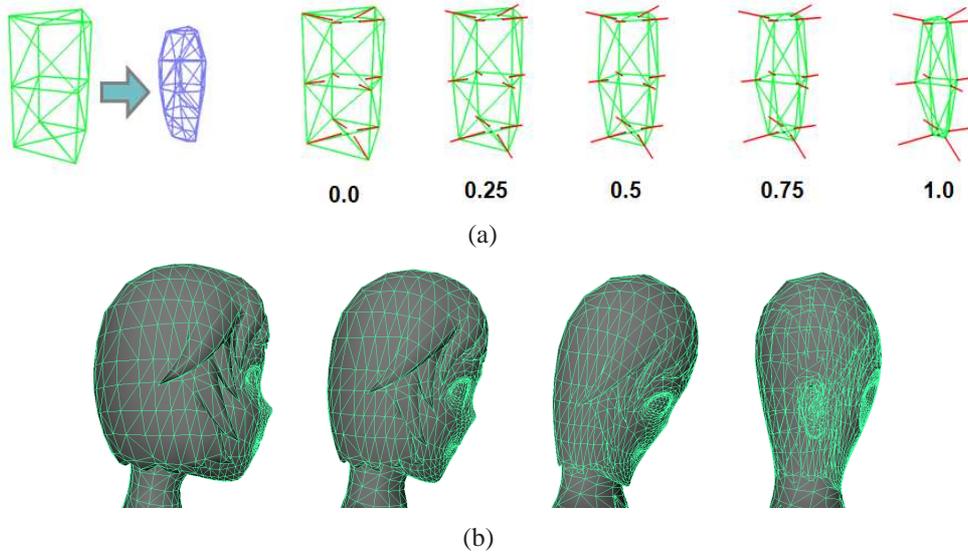


Figure 113: *The different stages of the projection of the BRep mesh to the FRRep shape of the “stand-in”:* (a) *Simplified example* (b) *Projection of a character’s head to the appropriate “stand-in”.*

simple gradient descent method or other more advanced methods. After the projection step, every vertex is assigned an offset to its position on the surface of a “stand-in” as well as the resulting normal vector. Then we use this information to perform the local deformation of the skinned mesh; i.e. we adjust the vertex positions in bind pose space and we apply a time-dependent skinning deformation to the mesh vertices at the same time. The same applies to the per-vertex normals:

$$p'_i(t) = \sum_{j=1}^N w_i^j \cdot \mathbf{M}_j(t) \cdot (f_\alpha(p_i, p_i + dp_i, t))$$

$$n'_i(t) = \sum_{j=1}^N w_i^j \cdot \mathbf{M}_j^{-1}(t) \cdot (g_\alpha(n_i, \tilde{n}_i, t))$$

where w_i^j is the weight of the j -th joint of the skeleton deformation, $\mathbf{M}_j(t)$ is the transformation of the j -th joint at time instance t , f_α is the interpolation function used to perform a smooth transition from the initial point p_i to the deformed point $p_i + dp_i$ over time, n_i is the normals associated with the i -th vertex and \tilde{n}_i is the normal at the surface of the “stand-in” at point $p_i + dp_i$. When the positions and normals of all the vertices are aligned with the

“stand-in” we switch from the polygonal object to the FRep object. This is illustrated in fig. 113a. Green represents the initial mesh object and purple is the approximate “stand-in”. The red vectors indicate the offset by which each vertex needs to be translated in order to be aligned with the “stand-in”.

In step **2**, we have FRep approximations of both animated meshes and we can employ different methods to generate the intermediate shapes. This method can be a straightforward FRep metamorphosis, a space-time blending or a complex user-controlled transition employing the skeletons defining the convolution surfaces inside the F-Cells. It is important to note that this transition can be evaluated automatically or can be defined by the artist. The result of this metamorphosis is an FRep object approximating the animated destination mesh.

In step **3**, we apply an inverse deformation to that applied in step **1**. Since we perform the metamorphosis using FReps all topological changes are handled automatically and we do not need to specify any additional constraints on the topologies of the original meshes.

All these steps are reflected in different IC instances used for the description of this dynamic hybrid model (see fig. 112). The K_1 IC instance reflects the state of the model when only a BRep mesh is present. This is a simple animation sequence, where the mesh is deformed over time using a skeletal deformation. In K_2 we perform a projection of the animated BRep mesh to its FRep “stand-in”. We still apply a skeletal deformation to the mesh and deform it in order to align it with the F-Cell containing the “stand-in”. Instance K_3 involves only F-Cells and the skeletons used for the deformation of both meshes. There are different options that can be used for the definition of the transition between these F-Cells. Instance K_4 is similar to K_2 , but here we apply the projection deformation in a reverse order. At the first time instance when this IC instance continues to be active, the resulting F-Cell is aligned with the animated and deformed destination BRep mesh. At the last time instance before this IC instance is invalidated, the animated BRep mesh is only deformed by the skeletal deformation and is aligned with the originally provided animated mesh (i.e. the animated BRep mesh is deformed to be aligned with the originally provided destination mesh). In K_5 the IC instance con-

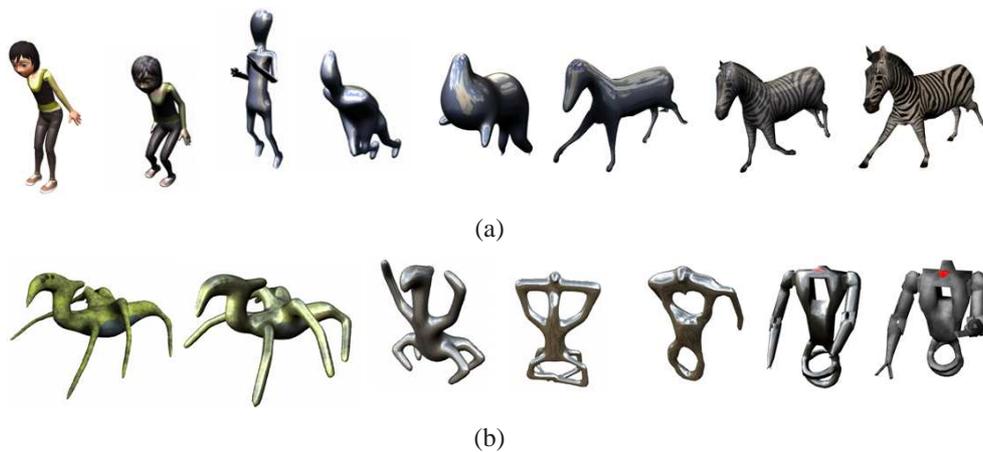


Figure 114: *The controlled metamorphosis of animated meshes: (a) Jumping girl to a running zebra metamorphosis (b) Crawling monster to a levitating robot metamorphosis.*

tains the destination animated mesh without any additional IC cells. At this point the controlled metamorphosis process is over.

We also perform smooth texture blending during the transition steps through a simple blend of the original texture and a “cubemap” texture using the up-to-date normal information (see fig. 114). Additionally, the transition between the animated mesh and the stand-in can be controlled by a “weight-map”, indicating which parts of the mesh have to be projected / unprojected first. This allows the user to deform the meshes in a non-uniform way (e.g. to collapse the head of the zebra before its tail).

This hybrid model can be defined interactively using our approach described in section 5.7. The interactive definition of a model is important for animators. As it provides them with a powerful tool for the creation of a transition effect which can be practically impossible to generate in currently existing BRep packages. The non-uniform time sampling feature of the IC framework²⁴ can be employed, at the intermediate transition phase, in order to improve the continuity of the metamorphosis sequence. Finally, the transition can be fine-tuned in near-real time and the resulting sequence can be rendered in real-time on the GPU (Kravtsov *et al.*, 2010c).

²⁴This subject was discussed in section 5.3.4

5.5.3 Conclusions

In this section we have described a new approach to the generation of metamorphosis sequences between two animated meshes using a hybrid model. This hybrid model incorporates both BReps and FReps, allowing us to easily switch between these representations depending on our needs. Unlike some of the existing approaches to metamorphosis between static meshes²⁵ our approach provides an additional degree of freedom to artists, furnishing them with tools allowing them to control the metamorphosis process. We believe that the incorporation of techniques such as this and the underlying hybrid modelling technology into existing and forthcoming modelling software and games engines will greatly enhance the ability of artists to generate complex models and animations.

5.6 The “Andyhausen” experiment

In this section we present an example model demonstrating some of the main features of the dynamic IC framework.

5.6.1 Model overview

The model consists of (see fig. 115):

- An “Egg” consisting of the “Eggshell” and the “egg contents”(see fig. 116). The “Egg contents” consists of two different volumetric space partitions. Each space partition represents the different materials of the “Egg’s” internal structure (i.e., the egg white and the egg yolk).
- A handle, a barrel and a piece of rope. One end of this rope is attached to the “Egg” and its other end is attached to a barrel that is connected to the handle. Rotating the handle causes the barrel to rotate. This in turn causes the “Egg” to either be lifted or lowered.

²⁵The problem of metamorphosis between animated meshes was not addressed in literature before.

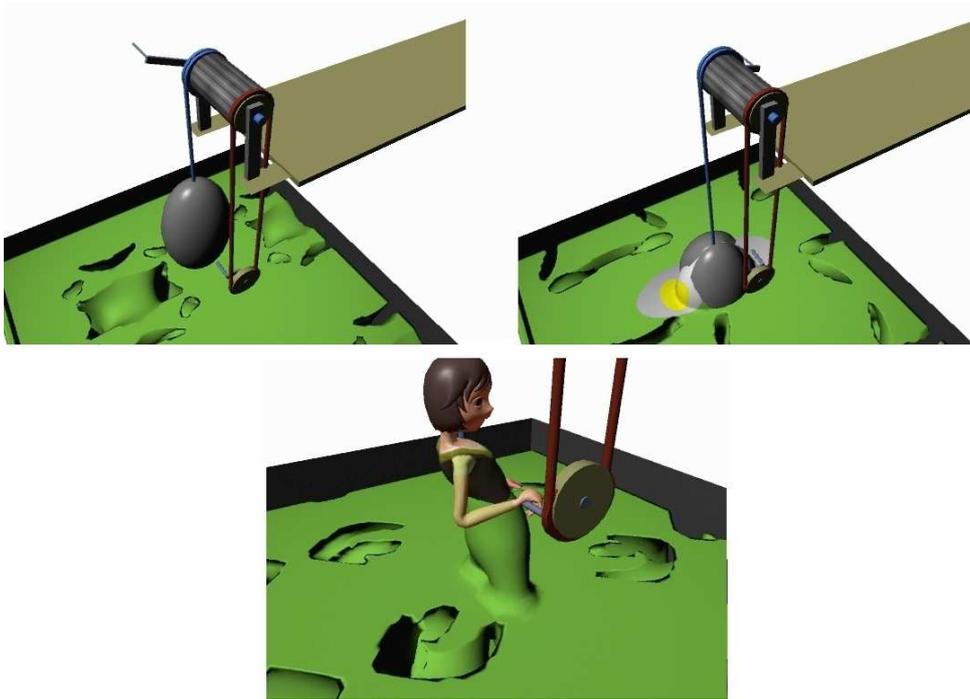


Figure 115: *The different states of the model.*

- A second handle that is connected to the first handle. Whenever one of the handles is rotated the other one is also rotated automatically.
- A liquid substance in which the “Egg” is being lowered.
- A girl called “Andy” (represented as a mesh) and her embedded stand-in convolution surface see section 5.4.

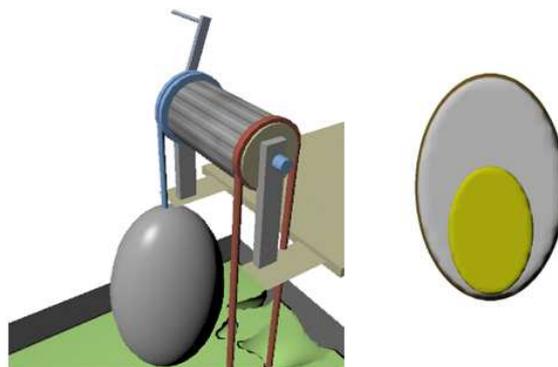


Figure 116: *The cells initially present in the IC.*

The dynamic IC has three instances (see fig. 115):

- First the handle is rotated, which results in the “Egg” being lowered into

the liquid until it touches its surface (see fig. 115 left). At that moment an intersection event between the liquid and the “Egg” is generated, which results in the transition to the next IC instance.

- When the eggshell touches the liquid it starts to crack, which results in the contents of the “Egg” being spilled into the liquid (see fig. 115 right).
- The mixture of the liquid and the egg contents leads to the creation of the “Andy” character, who then reaches out of the liquid to grab the second handle (see fig. 115 bottom). The movements of this character are defined using keyframed animation. The movements of the embedded stand-in, represented by an F-Cell, are synchronised with the movements of the character. Once the girl has grabbed the handle, she starts rotating it in order to pull herself out of the liquid.

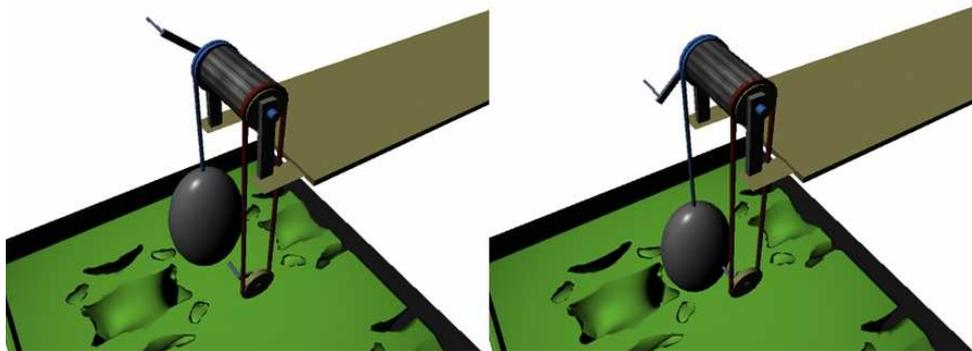


Figure 117: *The first IC instance.*

We assume that the “Egg” object is an implicit complex that has some internal structure (e.g., eggshell and egg contents with different types of attributes defining the yolk and the egg white²⁶). The ‘Egg’ could be defined outside this scene and we could just “insert” it into the current IC. This can be done through a union operation between the ICs. Additionally, we introduce a set of new dependency relations between the “Egg” and some other cells in this scene. Intermediate phases of the first instance of the IC are shown in fig. 117. The topological and dependency relations of this IC instance are shown in figures 118, 119 and 120 respectively.

²⁶In fact, it might have a skeleton, a convolution and a mesh, but we do not reflect these in the first instance of the complex, as they come into play only in the second instance.

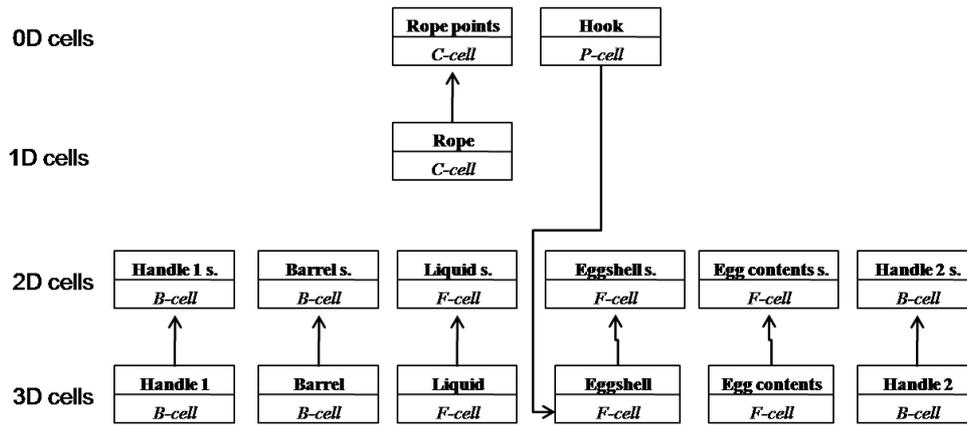


Figure 118: The boundary relations of the first IC instance.

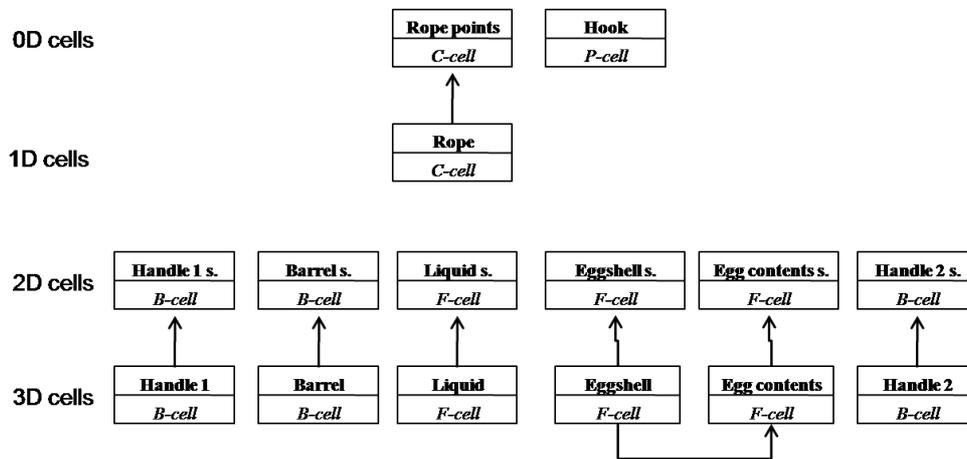


Figure 119: The containment relations of the first IC instance.

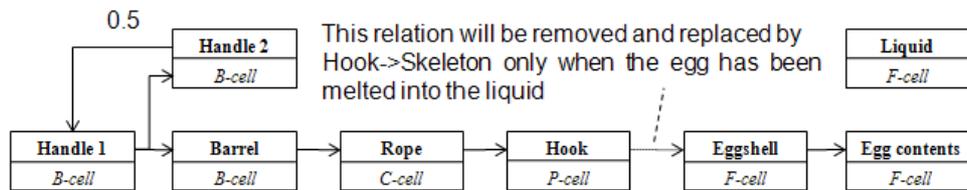


Figure 120: The dependency relations of the first IC instance.

The bi-directional dependency between the handles means that they should always be rotated by the same angle ²⁷. In this model we wish to establish the bi-directional dependency relation between the two handles. This means

²⁷NB A brief note regarding bi-directional and cyclic dependencies: At the start of the timeframe all dependency relations and cells are marked as invalid. When we update a cell it is marked as valid. After that we update its dependent cells. When both dependent and master cells are marked as valid, the dependency relation between them is also marked as valid. The update is complete when all relations and cells have been validated.

that we first update **handle1** (according to the priority of the dependency as defined in 4.4), then its dependent cells (**barrel** and **handle2**). Then, **handle2** issues an update of **handle1** again (because dependency between **handle2** and **handle1** is still invalid). **Handle1** then updates its parameters according to the new parameters of **handle2**. The dependency between **handle2** and **handle1** is now validated. A re-evaluation of barrel is not issued because the dependency relation between **barrel** and **handle1** is already marked as valid.

Here we must emphasise the fact that the “Egg” can actually be represented as an IC. The “Egg” has some internal structure (reflected in the second instance of the IC), but the user is not concerned with this at this stage. The user simply adds the IC to the model (e.g., through a union operation between ICs). The framework in its turn retrieves all the cells of the IC and merges them in a new IC. This allows the user to work with smaller models and tune them independently, thus providing a way for the creation of modular components that can later be interchanged and composed in a more complex model. For instance, the resulting IC can also be integrated into another implicit complex or some cells can be arranged into a sub-complex that can be exported/added to some library as an independent implicit complex.

5.6.2 Model description

We present the description of the IC using the high level declarative style definition introduced in section 4.3.1. Following the previously introduced methodology, first we introduce the descriptions of the cells that will be used in a number of the instances of the IC (see section 4.3):

```
CELLS {
  Handle1 {
    type = B-CELL;
    shape = loadMesh(...);
    dim = 3D;
    domain = {...};
    //the set of parameters/properties describing this cell and
    //their initial values
    parameters {
      omega : REAL(0.0); // angular velocity
      alpha : REAL(0.0); // angular rotation
    }
  }
  REACTIONS {
    update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
    {
```

```

        rotation.x = alpha; // update angle
    }
}
}
// copy the description from Handle1 (all params and evaluation
procedures
Handle2 <- Handle1 {
    REACTIONS {
        // customise the behaviour (though in this example we
        // don't need to)
        update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
        {
            rotation.x = alpha; // update angle
        }
    }
}
Barrel {
    type = B-CELL;
    shape = loadMesh(...);
    dim = 3D;
    domain = {...};
    // set of parameters/properties describing this cell and
    // their initial values
    parameters {
        radius : REAL(...); // barrel radius
        alpha : REAL(...); // barrel rotation angle
        distTravelled : REAL(...); // distance travelled
    }
    REACTIONS {
        update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
        {
            distTravelled = alpha * radius;
        }
    }
}
Rope {
    type = C-CELL;
    shape = LINE_SEGMENT(...), ARC(...);
    dim = 1D;
    domain = {...};
    // the set of parameters/properties describing this cell and
    // their initial values
    parameters {
        length : REAL(...); // the overall length of the rope
        inactiveLength : REAL(...); // the dynamic change of the
            // length of the rope
        curLength : REAL(...); // the length of the rope from barrel
        endPoint : REAL(...); // end point of the rope
    }
    REACTIONS {
        update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
        {
            curLength = length - inactiveLength;
            // the current translation of this cell:
            VECTOR3 trans = getTranslation();
            endPoint = VECTOR3( trans.x,trans.y-curLength,trans.z);
        }
    }
}

```

```

    }
  }
  Hook {
    type = P-CELL;
    shape = POINT(...);
    dim = 0D;
    domain = {...};
    // no additional parameters as the dependency for the transform
    // parameters is defined outside
  }
  Eggshell {
    type = F-CELL;
    shape = frep::TREE(...);
    dim = 3D;
    domain = {...};
    // the set of parameters/properties describing this cell
    // and their initial values
    parameters {
      // how quickly the egg is being damaged
      damageRate : REAL(...);
      // how much the shell has been damaged
      damageState : REAL(...);
    }
  }
  Egg_contents {
    type = F-CELL;
    shape = frep::TREE(...);
    dim = 3D;
    domain = {...};
    // no additional parameters as the dependency for
    // transform parameters is defined outside
  }
  Liquid {
    type = F-CELL;
    shape = frep::TREE(...);
    dim = 3D;
    domain = {...};
    REACTIONS {
      update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
      {
        // change the phase of the noise or perform some other
        // time-dependent action
        :
      }
    }
  }
  Melting_egg {
    type = T-CELL;
    // this cell is created using the blend between two existing
cells
    shape = IC::blendCells(Liquid, Egg_contents, ...);
    dim = 3D;
    domain = {...};
    parameters {
      // how strongly the egg contents blend with the liquid
      blendRate : REAL();
    }
  }

```

```

REACTIONS {
  update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
  {
    // change blendRate
    :
  }
} // Melting_egg
} // CELLS

```

The description of the IC template and the IC instances are provided in Appendix C.

After describing the first instance we introduce the second instance (the egg touching the surface, the shell starting to crack/erode, while the contents of the egg start blending with the liquid). The topological and dependency relations of this instance are shown in figures 121, 122 and 123 respectively.

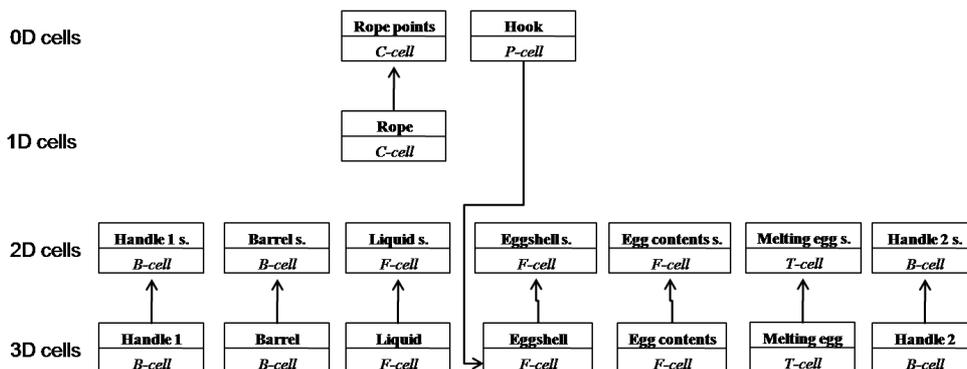


Figure 121: The boundary relations of the second IC instance.

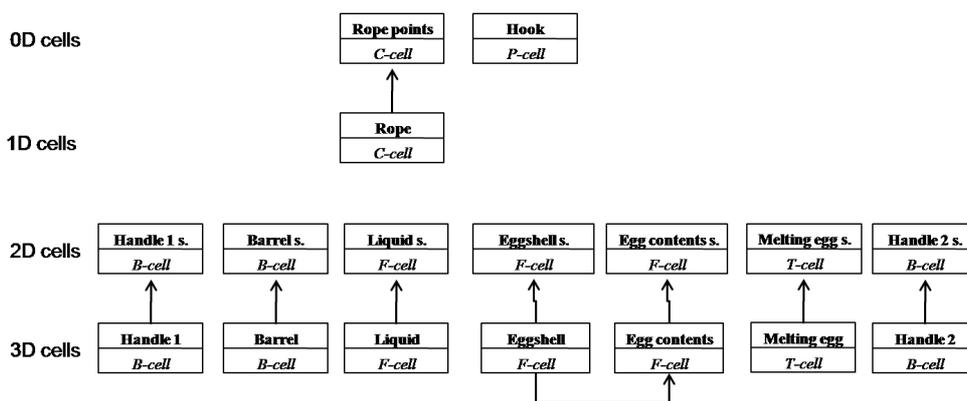


Figure 122: The containment relations of the second IC instance.

At the lowest point one of the Andy’s hands comes out of the liquid substance, grabs the handle and starts rotating it (see fig. 124). This action leads

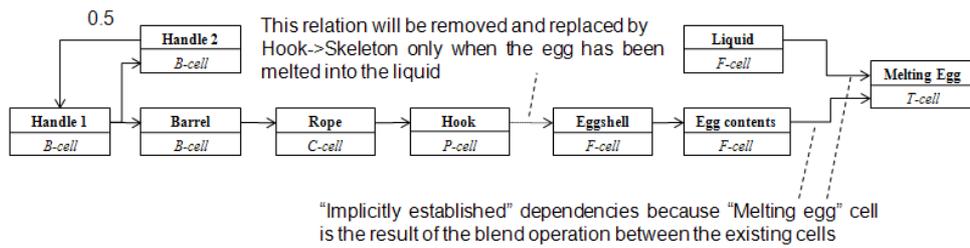


Figure 123: The dependency relations of the second IC instance.

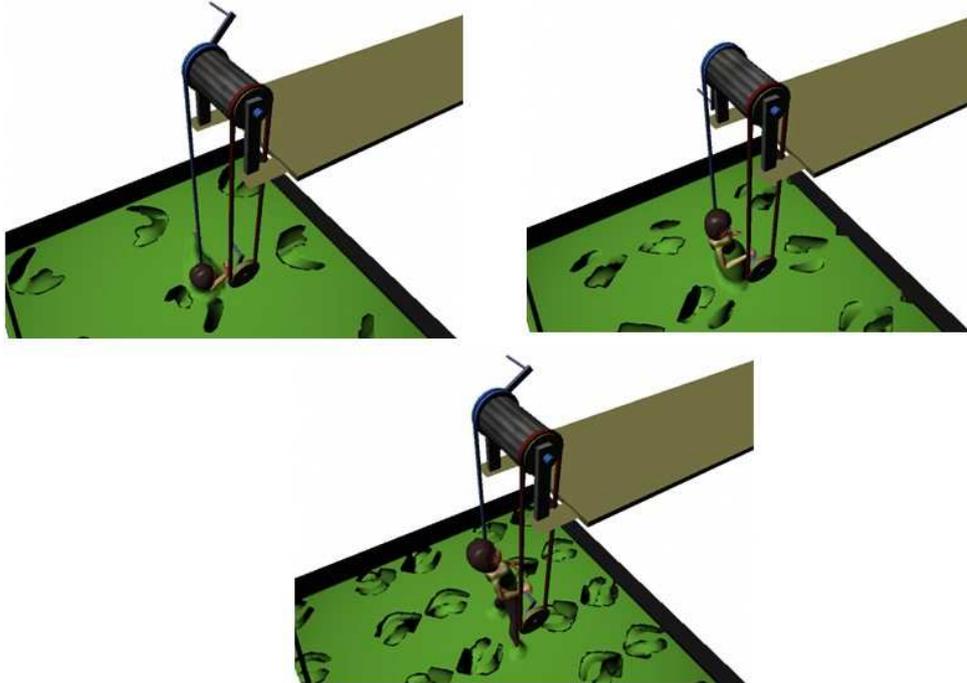


Figure 124: The third IC instance of the model.

to her rising from the liquid, since the handle is connected to the barrel via a belt. The movement of the belt leads to the rotation of the barrel in the opposite direction. There is a “Liquid Andy” cell, which is an FCell approximating Andy that is driven by the same skeleton that is used to animate Andy. “Liquid Andy” allows us to model the interaction between the animated BRep model and the liquid, i.e., the viscous behaviour of the liquid “sticking” to Andy. This state of our dynamic model is reflected in the third IC instance. This is almost the same as the second one. Only we now have a new circular dependency caused by the dependency relation between the cell **Andy** and **handle2**. The idea here is that we assign a higher priority to the dependency between the cells **Andy** and **handle2** than the priority we assign to the depen-

dependency between **handle2** and **handle1**. Thus the update of **handle1** according to its dependency on **handle2** will be the last evaluation performed in the IC²⁸. This means that **handle1** will be synchronised with the state of **handle2**. The cell “Eggshell” no longer exists and there is no dependency between the cell **egg liquid** and the **hook**. We add a dependency between the **hook** and the skeleton instead.

The boundary and dependency relations of this instance are shown in figures 125 and 126 respectively.

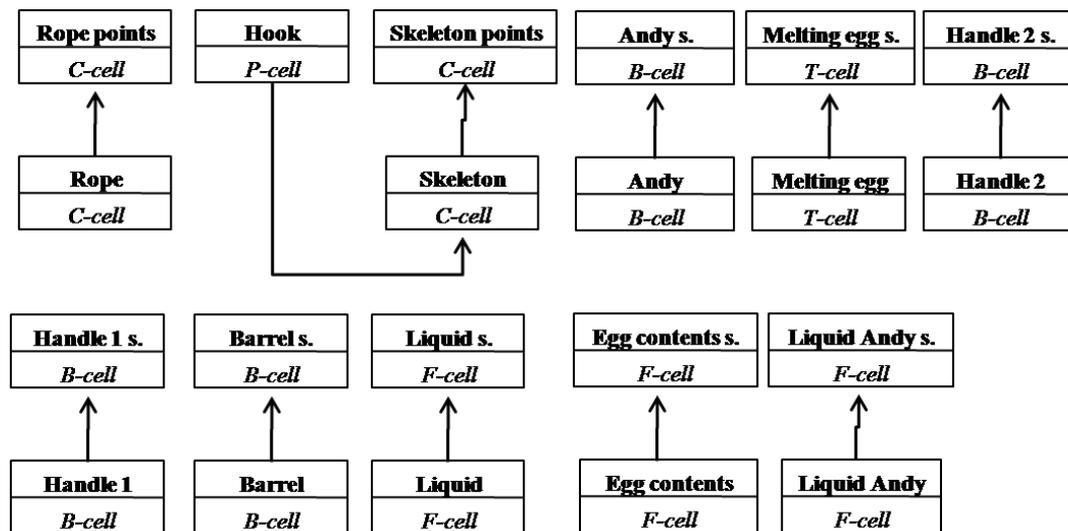


Figure 125: The boundary relations of the third IC instance.

Additional modifications of “Andy’s” stand-in can also be introduced (e.g. a metamorphosis of her limbs from a mermaid to a human lower body or something similar). The full model definition allows us to generate an animation sequence reflecting all the intermediate states of the model (see fig. 115).

5.6.3 Conclusions

This example model demonstrates the following features of the dynamic IC framework:

²⁸i.e. all dependencies with higher priorities are validated first. This allows the user to have some control over the sequence of the performed actions.

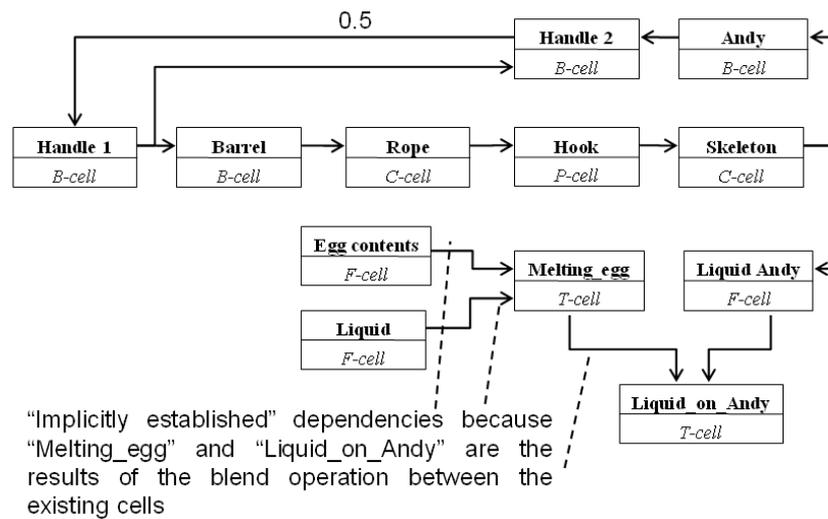


Figure 126: *The dependency relations of the third IC instance.*

1. The use of a model that utilises hybrid representations including FReps (the Egg, the liquid and the liquid mixed with the egg contents) and B-Reps (Andy and the barrel).
2. The use of multidimensional cells (0D hook, 1D rope, 2D Andy surface model, volumetric 3D egg, etc.).
3. The representation of volumetric heterogeneous structures and the relations between them (i.e. the egg contents located inside the eggshell).
4. The unified dependencies between the cells with the ability to control the order of their evaluation (i.e. the bi-directional dependency relation between the handles and the composite dependency cycle between the cells).
5. The volumetric attributes assigned to a subset of the cells present in the model (i.e. the volumetric heterogeneous material describing the egg contents and its mixture with the liquid).

This example demonstrates that the dynamic IC framework allows us to exploit all the advantages of traditional animation techniques to define time-dependent hybrid models of heterogeneous objects.

5.7 An interactive modeller for the definition of volumetric hybrid models

It is apparent that the textual description of an IC-based model can easily become rather large. Besides, a textual description of the model is error-prone and requires certain technical skills from the modeller. This makes ICs less accessible to a wide group of artists who could benefit from the features provided by the dynamic IC framework. Thus, a simpler method of model definition will have to be provided, so that a wider audience could gain access to the capabilities of the IC and FRep frameworks. As a consequence, users of varying levels of ability would be given the opportunity to experiment with the framework and to create models of varying complexity. A custom GUI built around the concepts of ICs could be designed to provide a way to create and to manipulate certain aspects of a hybrid model in an easy way. A full-blown interactive modeller based on the concepts of ICs is a large-scale project and is outside of the scope of this thesis. But using the APIs described in sections 4.5 and 4.6 we are able to provide the user with a set of tools to define certain types of cells and their behaviours. A generic GUI of existing third party software packages could also be used to manipulate a hybrid model. In this case we need to provide an intermediate layer between the IC API and the specific API of the modelling application we wish to extend as was described in section 4.6.4.

5.7.1 General description

As a proof of concept we have chosen to incorporate FRep modelling capabilities into the Autodesk® Maya™ software package (Autodesk, 2011a). This package is widely used by professional artists for projects of varying complexities. It provides a set of tools for surface modelling, for rigging and both for the keyframed and the procedural animation coupled with physical modelling capabilities. Maya™ is a highly extensible platform and it provides a flexible way to develop custom plug-ins.

We have implemented a set of Maya™ plug-ins which can be used for the definition of an FRep tree of F-Cells.

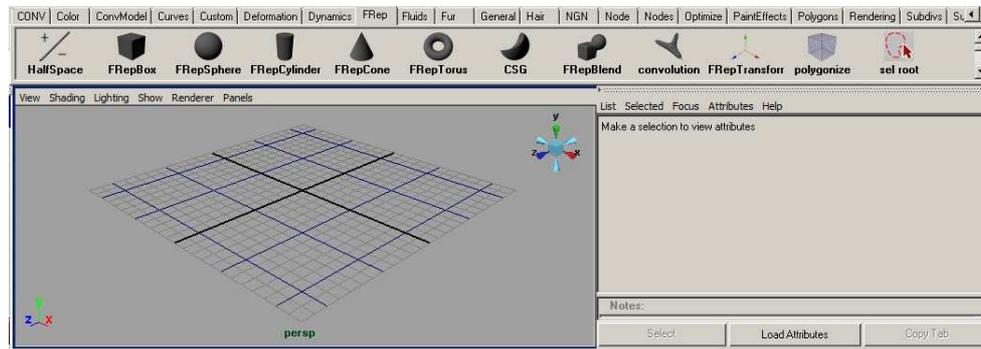


Figure 127: *The FRep shelf added to Maya.*

The user can manipulate the FRep model through a custom menu added to Maya (see fig. 127). A set of FRep primitives and operations are available to the user. Added FRep primitives are shown in Maya in the form of proxy wireframe objects which can be manipulated as any other Maya built-in object (see fig. 128). The FRep primitives are also shown in the Directed Acyclic Graph (DAG), which reflects all geometric objects present in the scene (see left of 128). This provides the user with the means of manipulating the model using visual metaphors. This means that the user is not required to have deep understanding of 3D geometry or FRep modelling. More experienced users still have access to the FRep tree, which can be modified on the fly using the Maya Dependency Graph (DG) as illustrated in fig. 129. The rendering of the proxy primitives can be disabled at will at any given time in order to gain better understanding of the intermediate shape (see figure 130). In general the model is modified through the parameters available for each FRep entity. Fig. 131 illustrates the set of parameters available for the blending union operation. The modification of any parameter results in the re-evaluation of the FRep model and in the subsequent rendering of the resulting shape. Additionally, the values of these parameters can be animated using the set of techniques available in Maya (see fig. 132). This allows the user to create complex time-variant FRep models in a relatively easy way. The produced time-dependent shapes can then be rendered using the tools available in Maya or be exported elsewhere. For instance, the produced animated models can be exported for real-time playback on the GPU (see detail in section 5.7.2).

Fig. 133 illustrates a model produced in Maya using our FRep plug-in. It is hard to estimate the time it would require to define this model through

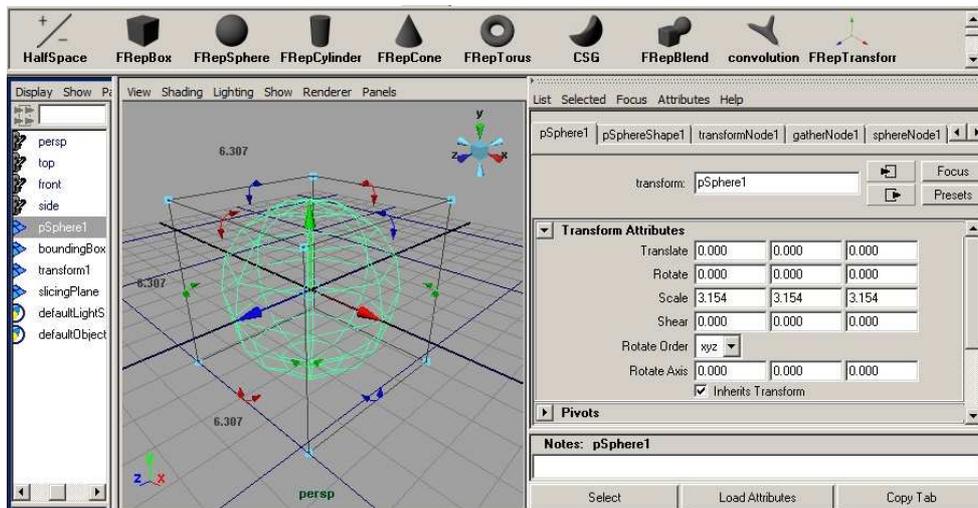


Figure 128: The FRep proxy object representing a solid sphere in Maya.

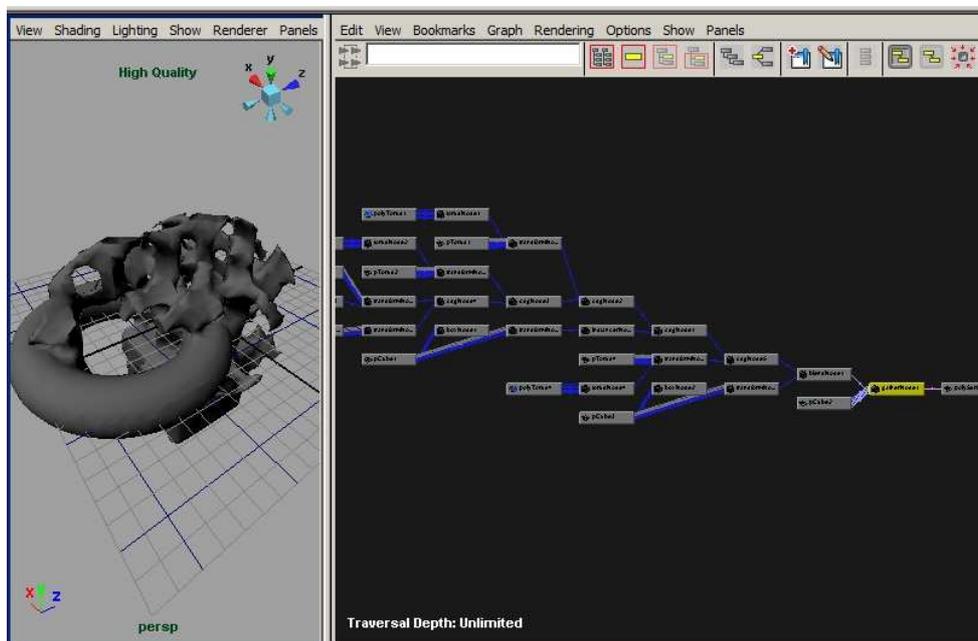


Figure 129: Constructive FRep tree shown using the Maya Dependency Graph or Hypergraph.

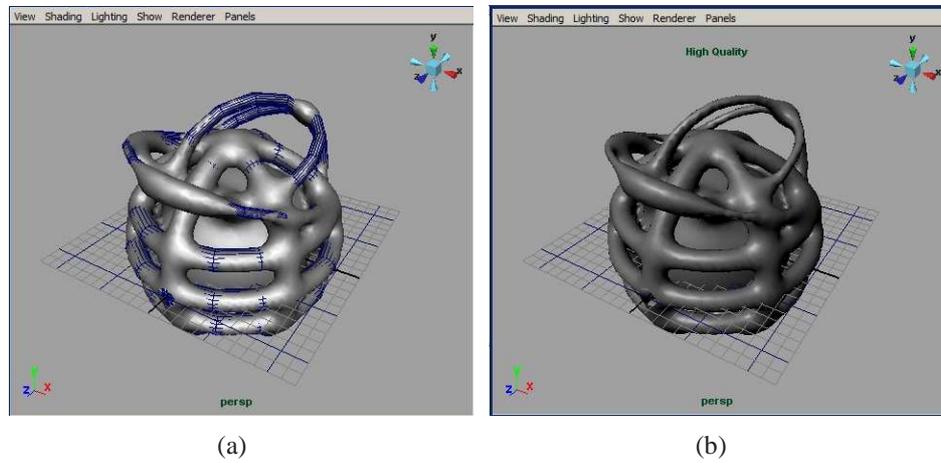


Figure 130: Two rendering options in Maya: (a) The rendering of proxy primitives is enabled, (b) The rendering of proxy primitives is disabled.

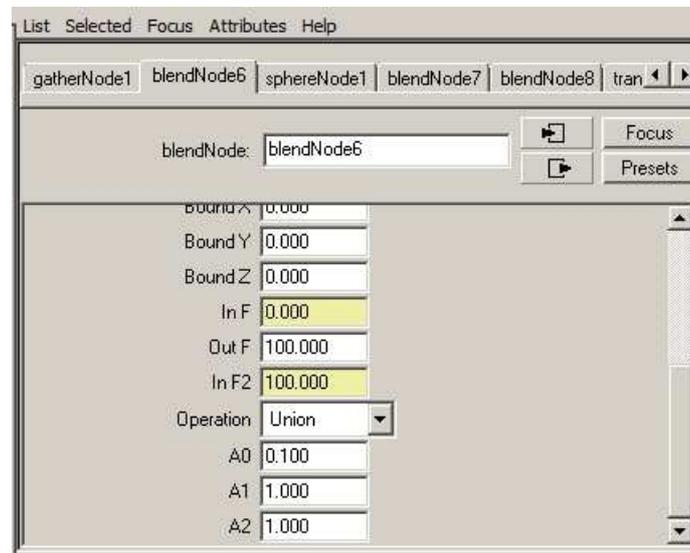


Figure 131: The parameters available for the blending union operation.

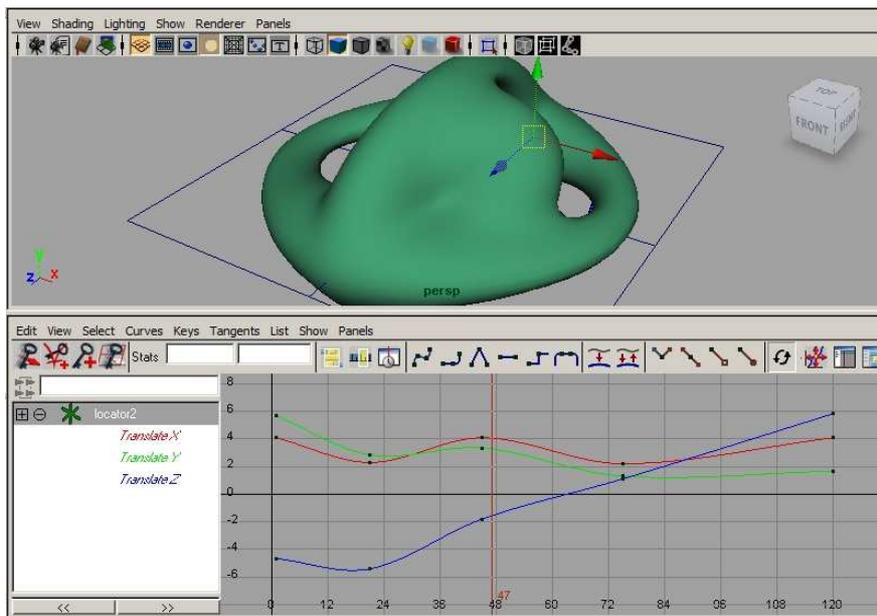


Figure 132: Animating the available parameters of FRep entities using the Maya animation tools.

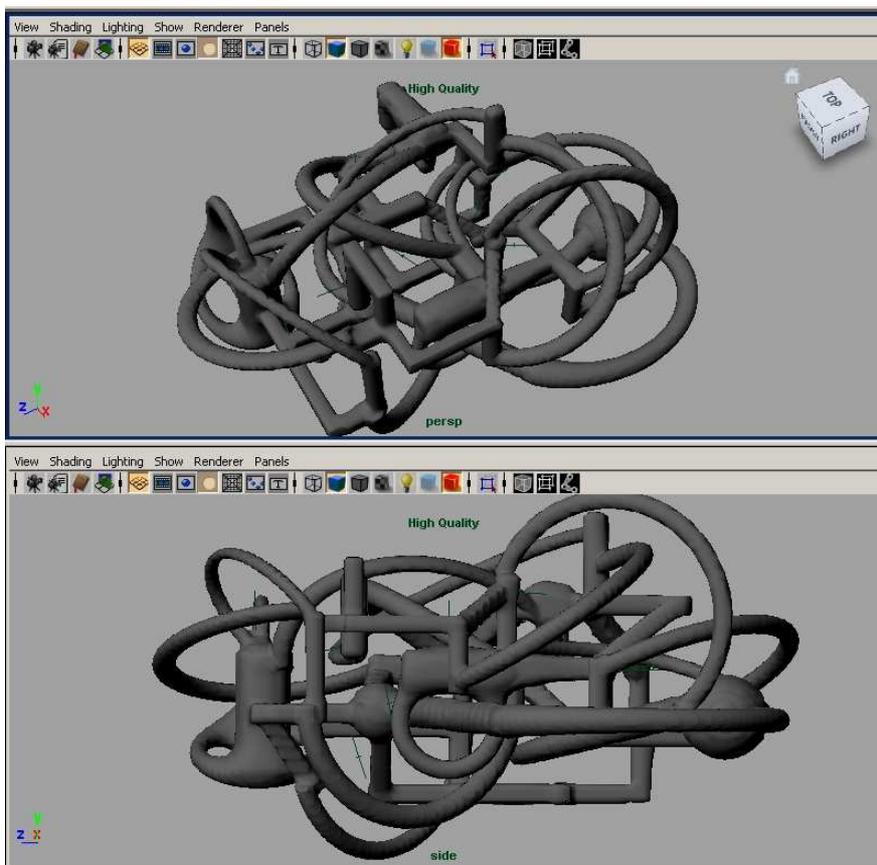


Figure 133: Example of an FRep model created within Maya shown from different angles.

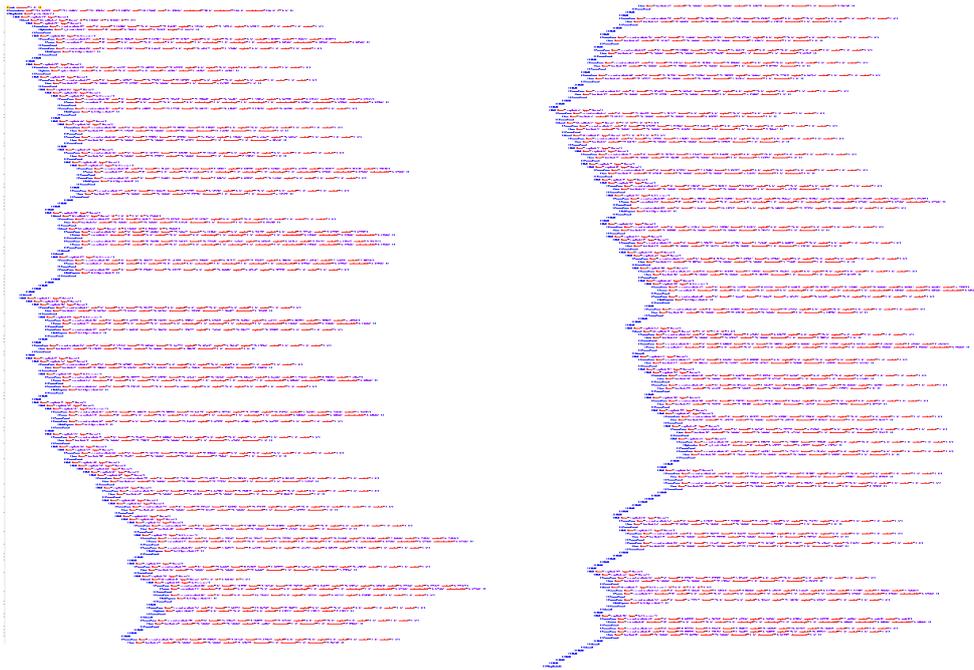


Figure 134: *An overview of the XML description of the model shown in figure 133.*

textual description. The definition of this FRep model in our XML format is depicted in fig 134. This FRep model consists of 250 FRep entities, 84 of which are FRep primitives (see fig. 135). This particular FRep model was created using our Maya plug-in in approximately three hours. This model was defined iteratively while using low-resolution²⁹ rendering in order to get a visual impression of the resulting shape on the fly. The final result was rendered with a high-resolution discretisation of the model. This required a lengthy evaluation procedure. It is important to note that there is no need to perform polygonisation of the entire model during the modelling process. The user can adjust the bounding volume around the region of interest during any particular stage of a modelling operation. Only the selected region of space would then be updated when any modification to the FRep model is performed. This can help us reduce significantly the evaluation times during the modelling process.

We also provide additional tools for 2D modelling. The user can retrieve an image of the scalar field of slices of a specific FRep object. Additionally a

²⁹In the current implementation we use polygonisation (see section 5.7.2) in order to render the resulting shapes.



Figure 135: *The FRep tree of the model shown in figure 133.*

set of slices can be generated for rendering of space-time models (see sections 3.4.2 and 5.3) as was shown in figures 93 and 95.

We also provide the user with a set of tools for real-time and off-line rendering of the produced models using procedurally-based texturing techniques. These tools are crucial in the world of computer animation and visual effects, where additional detail can be added to a three-dimensional object through a set of high-resolution 2D images. Figure 136 illustrates a set of examples rendered using some available textures. It is important to note that the user is not required to perform traditional parametrisation of the geometric shape in order to create the UV mapping commonly required for the texturing of BRep objects.

The majority of the examples and illustrations provided in this thesis were produced using Maya extended by our plug-ins. Our Maya FRep modeller has evolved into a powerful modelling system which can be used by users of varying abilities.

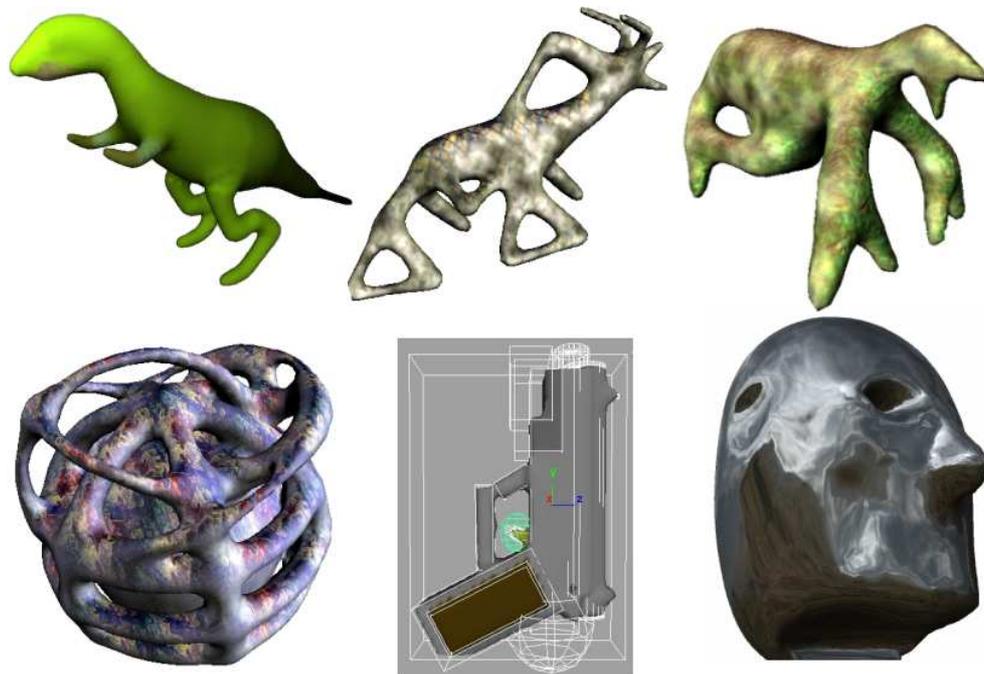


Figure 136: *Examples of volumetric models produced using our modeller.*

5.7.2 Implementation overview

In our current work the Dependency Graph (DG) of Maya™ is used in order to build an FRep tree for F-Cells. This is needed because Maya™ is mostly oriented towards BRep modelling, while we want to provide the user with a similar functionality for the definition of F-Cells. The DG allows us to establish connections between arbitrary nodes via their attributes. We need to provide a set of Dependency Graph nodes representing the entire set of FRep entities (see fig. 62 and 63). These custom DG nodes implement an intermediate layer, as was discussed in section 4.6.4. The parameters of each entity available to the user need to be exposed separately through Maya™ attributes. The DG is only used to provide topological information for the FRep API. An intermediate layer is used to retrieve the graph information from Maya™ in order to set up the tree using the FRep API. All operations are then performed within the FRep API and Maya™ is only used to render the final results (see fig. 136). Rendering is performed through the polygonisation of the FRep object at a specified iso-level. The user can choose between the regular Marching Cubes algorithm (Lorensen and Cline, 1987) or one of the feature-preserving surface extraction algorithms (Kobbelt *et al.*,

2001) available, as demonstrated in fig. 137. We have also implemented an adaptive iterative polygonisation method which allows us to significantly reduce the time required for surface extraction.

The implementation of the system described in section 4.6.6 significantly simplifies the process of adding new FRep entities to our Maya plug-in. An intermediate translator produces the code necessary for the basic integration of our custom entities into Maya using the Maya API. So that the developer concentrates on the implementation of the actual entities within our APIs. The high-level description of the entities is then used to produce Maya specific code, which is not directly related to the FRep evaluation procedure.

This allows us to solve modelling and animation problems which are next to impossible to overcome using the available set of tools present in this package. The produced models can be exported to HyperFun or to the custom XML format described in section 4.6.5. The models serialised as XML files can then be loaded in a standalone FRep viewer or be used for interchange with other applications extended with FRep modelling capabilities through the FRep API. Another interesting option is the export of the model to a CUDA kernel (see section 4.6.8) for its evaluation on a GPU. This allows us to significantly improve the performance of model evaluation, which is especially important for the rendering of animated models. We have provided details regarding the GPU implementation and its applications in (Pasko *et al.*, 2010; Kravtsov *et al.*, 2010b,c).

Finally, interactively defined F-Cells, animated B-Cells and C-Cells can easily be integrated into an IC-based model as outlined in section 4.6.7.

5.7.3 Future work

Above we have presented a description of the integration of our tools into the Maya animation package. Other popular packages currently available in the marketplace could also be extended in a similar manner. For instance, Softimage (Autodesk, 2011b) or Houdini (Side Effects Software, 2011) also have a node based dependency system. This means that the integration of the FRep and certain IC features could be done using a similar approach as the

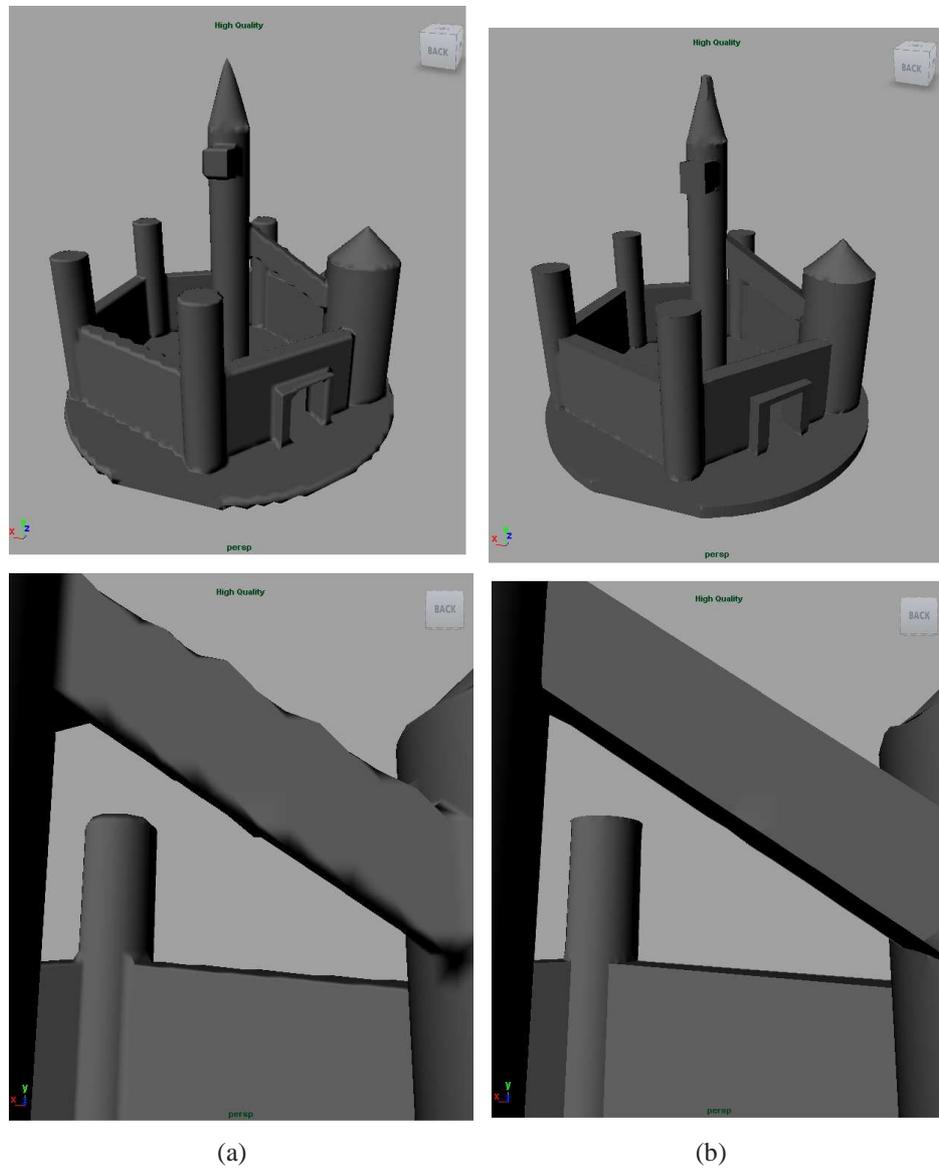


Figure 137: *The FRep castle model: (a) Using the regular Marching Cubes algorithm, (b) Using the Marching Cubes algorithm with post processing extracting sharp features.*

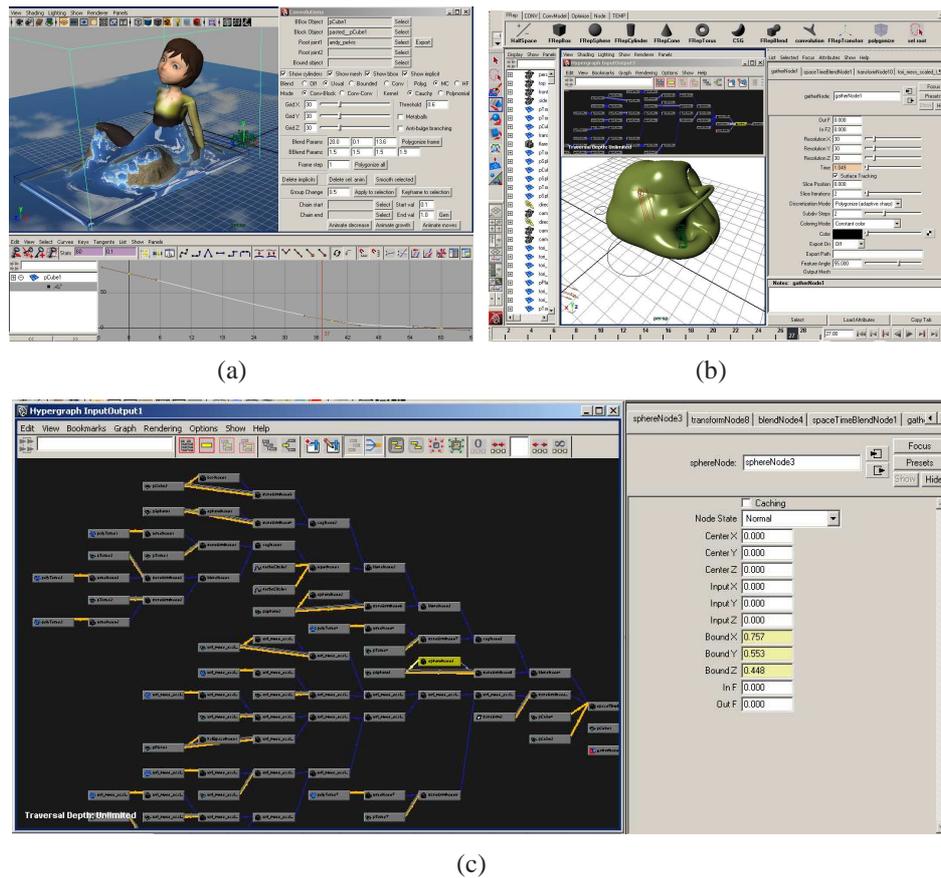


Figure 138: Screenshots of the working environment: (a) The definition of the “Stand-in” model, (b) The setup of the improved space-time blending model, (c) Exploring the FRep tree contained in the IC F-Cell used for space-time blending.

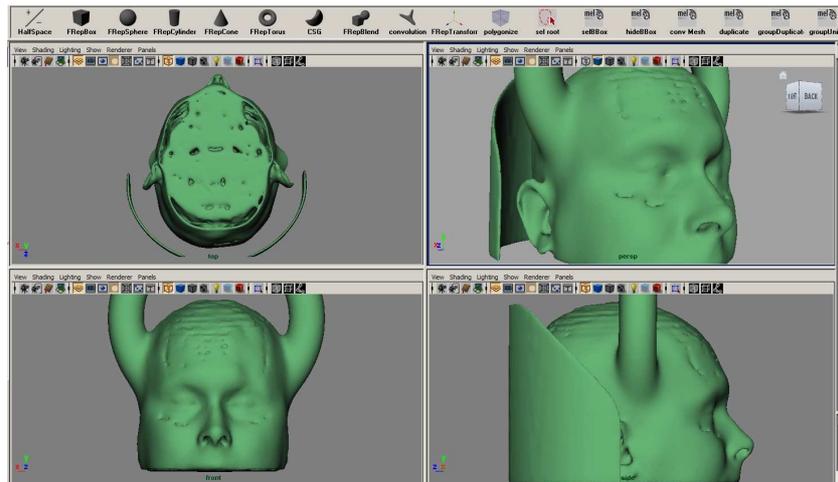


Figure 139: Scanned voxel data of a patient blended with FRep entities.

one we used for Maya. These packages have a large user base across different industries and their capabilities are constantly being improved.

The incorporation of the functionality provided by the IC API into existing software packages decreases the learning curve for the user. The user is free to produce an animation sequence in a way that he is accustomed to within a familiar software environment. Additionally, this allows us to integrate IC models into complex scenes created within existing packages. Our Maya plug-ins are currently used by the students at the National Centre for Computer Animation at Bournemouth University. In the future these tools can be used for medical applications (see figure 139) and 3D printing applications (see figure 140).

We plan to further improve this plug-in, as we believe that in the future our FRep tools will attract more users from both academia and industry.

5.8 Conclusions

In this chapter we have presented a number of examples and experiments of varying complexity. First, we presented two simple examples in order to illustrate the methodology of the dynamic hybrid model definition. After presenting these illustrative examples we have described a set of existing practical problems which were resolved using dynamic hybrid multi-dimensional



Figure 140: *Examples of rapidly prototyped (through 3D printing) FRep objects defined using the HyperFun package.*

models. We have presented an improved solution to the problem of metamorphosis between FRep objects using a mixed-dimensional dynamic model. Next, we have outlined our solution to the problem of modelling interactions between animated meshes and viscous objects using our hybrid model. As we have demonstrated, our approach of using the “stand-ins” can be extended to a set of other modelling problems, such as two-way hybrid character modelling and animation, and partial metamorphosis of animated characters. The “stand-ins” technique also allows us to model controlled metamorphosis between animated characters. The metamorphosis process involves models in different representations, so that we can benefit from the advantages of each of the utilised representations. The “Andyhaussen” experiment demonstrated all the advantages of the proposed dynamic IC framework. We were able to model interactions between multidimensional dynamic heterogeneous objects using a set of IC instances for the representation of our hybrid model. We have also presented a description of our prototype implementation of an interactive modeller that is used for the definition of complex dynamic models.

From the discussion in this chapter we can see that the hybrid modelling approach has already allowed us to solve a number of challenging problems. Future research in this direction promises to provide answers to a number of as yet unanswered questions. We strongly believe that the incorporation of the proposed techniques and techniques yet to be developed, that are based

on hybrid modelling concepts, into existing modelling software and game engines will greatly enhance the functionality of these systems. This will result in a new degree of creative freedom and will improve user experience in a number of application areas.

6 Conclusions and future work

First of all let us summarise the topics covered in this thesis.

In Chapter 1, we have presented a general overview of the subject area. This overview included the outline of existing approaches to the modelling of static and dynamic objects. We have also outlined a set of existing issues and challenges in this area.

In Chapter 2, we have presented related work and we have briefly described existing model representations. We discussed the advantages and disadvantages of the various representations together with their application areas. The Implicit Complexes (IC) framework was introduced as a common platform allowing us to incorporate all the representations within one hybrid model. We have provided a survey of the existing animation techniques and methods used for the definition of time-dependent models. The presentation of this preliminary material was essential in order to justify the necessity for the extension of the IC framework so that we could use it for the definition of time-variant heterogeneous objects.

In Chapter 3, we have introduced a new Dynamic IC framework. First, we have extended the previously available IC Framework through the new definitions of entities required for the description of a dynamic hybrid model. Secondly, we have introduced new notions allowing us to define the structural and the parametric states of the entire dynamic event-driven IC model. Additionally, we have introduced a set of extensions to the Constructive Hypervolume Framework (CHF). The CHF is based on the Function Representation (FRep) and is incorporated into the dynamic IC framework, playing an important part in the definition of multidimensional time-variant heterogeneous objects.

In Chapter 4, after the introduction of our new theoretical framework, we have described our approach to the system design and the implementation of the practical tools allowing us to work with dynamic IC models. We have mapped a set of notions available in the IC framework to a set of entities in an Object-Oriented Programming (OOP) style, thus making it easier for the user to map a conceptual dynamic hybrid model to a practical definition of

an IC-based model ready for evaluation. Additionally, we have presented a methodology, which can be used as a guide for the breakdown of a complex hybrid model into a set of components and a set of links between them. A new notation and a new high-level language designed for the definition of dynamic hybrid models was introduced in the same chapter. This language is built on top of the IC API, which allows us to perform the evaluation of the hybrid model. The IC API can be used by other applications, thus providing them with hybrid modelling capabilities. We have presented a breakdown of the steps required for the evaluation of the IC-based model and we have discussed the majority of the technically challenging aspects of the evaluation procedure. This information is necessary for the practical implementation of the framework. Finally, we have introduced a new FRep API, which is necessary for the integration of the Constructive Hypervolume Framework into the dynamic IC framework. This FRep API may also be used independently of the ICs, thus providing a wide range of users of varying abilities with a set of tools required for full-blown FRep modelling unavailable elsewhere.

In Chapter 5, after the definitions of theoretical and technical novelties, we have provided a set of practical solutions to a number of known problems. These include the hybrid modelling of a set of simple dynamic objects, the modelling of a multidimensional metamorphosing objects defined in the space-time domain and the physically-inspired modelling of the interactions between dynamic meshes and viscoelastic substances. We have also presented our solution to the complex problem of metamorphosis between animated meshes using our hybrid modelling approach. We have described a case study demonstrating a number of features of the IC framework. This case study features the interaction of hybrid multidimensional time-variant objects, which are defined together with a set of dynamic relations between them and their volumetric properties that are changing over time. Finally, we have described the set of tools we have developed through the integration of our APIs into existing commercial 3D modelling packages. These tools allow us to take advantage of both the available multi-core CPUs and GPUs, significantly improving the performance of the hybrid model evaluation process.

There is still ample room for the further improvement of the dynamic IC framework. More attention needs to be paid to concurrent interaction between

the IC entities and to the definition of complex event-driven dynamic models. Additional work is required for the further implementation of the IC API and the FRep API. We plan to finish the full implementation of the proposed language designed for the definition of dynamic IC models. We would also like to further investigate the visual aspects of hybrid modelling. A new set of tools built on top of the IC API should be made easily accessible to a set of users of varying abilities, through the combination of the proposed high-level model definition language and the set of visual metaphors available through the advanced GUI. We are also hoping to conduct an in-depth investigation of the advantages of dynamic IC models in a number of physical simulation applications.

We believe that the incorporation of the dynamic IC framework and the underlying hybrid modelling technology into existing and forthcoming modelling software and game engines will significantly enhance the functionality of these systems. This will also greatly enhance the model-building capability of the artistic community.

6.1 Contributions

In this thesis we have presented the results of our research into hybrid modelling of dynamic heterogeneous objects. Here we summarise a number of main contributions of this thesis:

- The introduction of a new Dynamic IC framework. This was initially achieved through the extension of the previously available IC Framework and through the subsequent introduction of the new definitions required for the description of a dynamic hybrid model. These newly introduced notions allowed us to define the structural and the parametric states of the entire dynamic IC model. The definition of the dynamic behaviour of the model was achieved through the combination of procedural time-dependent model definitions, based on event-driven dynamics, together with some widely-used traditional keyframe-based approaches. This provided us with the means of describing complex behaviours of a model in a relatively simple way.

- The introduction of a set of extensions to the Constructive Hypervolume Framework (CHF). These new extensions improved the capabilities of the CHF required for multidimensional time-variant heterogeneous object modelling.
- The introduction of a new high-level domain-specific language suitable for the definition of the introduced dynamic hybrid models. This domain-specific language allows us to define complex dynamic hybrid models in a relatively easy way using all the notions introduced for dynamic IC-based models. This was achieved through the mapping of a set of notions available in the IC framework to a set of entities in an Object-Oriented Programming (OOP) paradigm. The textual definition of the model allows us to directly map all theoretical concepts from the dynamic IC framework and from the CHF to a set of OOP-entities. This significantly simplifies the process of the mapping of a conceptual dynamic hybrid model to a practical definition of an IC-based model which can be evaluated.
- The design and the development of the software tools, namely the IC API, allows us to work with dynamic IC models. The IC API allows us to perform the actual evaluation of the dynamic hybrid model based on its valid description. We presented a discussion relating to the design and internal mechanisms of the IC API. The high-level language, described above, is based on the functionality introduced by the IC API. The IC API can also be used by other applications, thus providing them with hybrid modelling capabilities.
- The design and implementation of a new FRep API. This API is necessary for the integration of the FReps and of the CHF into the dynamic IC framework. The FRep API allows us to realise the full potential of the Function Representation unlike previously available FRep tools. This FRep API may also be used independently of the ICs, thus providing a wide range of users of varying abilities with a set of tools required for full-blown FRep modelling not previously available. It is important to note that we have also presented a detailed description of the process required in order to introduce, with relative ease, further improvements

and extensions to the FRep API, independently of the underlying software or hardware platforms being used.

- The incorporation of the IC and FRep modelling features into the popular animation and modelling software package Autodesk® Maya™. Apart from providing the user with a set of new features integrated into a familiar environment, we presented a detailed description of the generic system that can be used for the integration of our APIs into third-party software packages. This approach simplifies the integration process significantly, thus allowing us to perform automatically the bulk of the integration and maintenance procedures.
- The introduction of our novel approach based on the concept of animated implicit “stand-ins”. The description of our method included the approximation procedures, the method of synchronisation and the physically-inspired emulation of viscoelastic behaviours by using geometric methods alone. The proposed approach allows us to solve a set of known problems in computer graphics. These problems include modelling of the interactions between animated meshes and liquid substances, partial and full metamorphosis of animated characters.
- The description of hybrid models which illustrated the advantages of our dynamic IC framework. Our experiments with hybrid models demonstrated the possibility of the interaction of hybrid multidimensional time-variant objects, defined within one hybrid model. Relations between these objects and their volumetric properties are allowed to change over time.
- The extensions introduced to the space-time blending technique, which provide the user with a higher level of control over the metamorphosis process through the inclusion of additional means of control.
- The implementation and the discussion of current and future improvements in performance required for faster evaluation of dynamic IC-based hybrid models and of pure CHF models using both the currently available CPUs and the GPUs.

6.2 **Future work**

There is still ample room for the further improvement of the dynamic IC framework.

First of all, more attention needs to be paid to the concurrent interaction between IC entities and to the definition of complex event-driven dynamic models. Further investigation needs to take place regarding the integration of simulation capabilities into the dynamic IC framework. A deeper understanding of the simulation requirements, specific to various application areas, may require the introduction of new concepts and notions into the framework. At the moment it is obvious that even simple geometric classification operations, such as intersection tests or point membership classifications, within a hybrid model require additional consideration because of the complexity and efficiency of these tasks when dealing with hybrid models. Robust classification operations can significantly simplify the definition and the validation of dynamic hybrid models.

Additional work is required for a more complete implementation of both the IC API and the FRep API. We plan to conclude the full implementation of the proposed language designed for the definition of dynamic IC models and to further extend the APIs and the DSLs built on top of those taking into account the modifications of the theoretical framework.

We would also like to further investigate the visual aspects of hybrid modelling. A new set of tools built on top of the IC API should be made easily accessible to the users of varying abilities, through the combination of the proposed high-level model definition language and the set of visual metaphors available through the advanced GUI.

It is also desirable to develop more experimental models and real-world case studies that are defined using the IC framework. We are also hoping to conduct an in-depth investigation of the advantages of dynamic IC models in a number of existing physical simulation applications.

Another important aspect of hybrid modelling is the rendering of the resulting model. Although we have described a number of available and proposed techniques both on the CPUs and on the GPUs, there is still a lot of

room for improvement. The efficient rendering of hybrid models in shorter times allows for faster iterations and for higher accessibility of the dynamic IC features to a wider audience. This means that robust rendering has to be one of the priorities in further research.

An important aspect of static and dynamic volumetric modelling is that of volumetric attributes. Due to time constraints, we have had to limit ourselves to a rather cursory examination of the Constructive Hypervolume Framework and of its advantages and of its applications. In the future more attention needs to be paid to arbitrary volumetric attributes within the IC framework. This is crucial for physical simulation applications and for the texturing capabilities of both static and dynamic shape modelling.

In conclusion, we believe that the incorporation of the dynamic IC framework and the underlying hybrid modelling technology into existing and forthcoming modelling software and game engines will significantly enhance the functionality of these systems. This will also greatly enhance the model-building capability of both the scientific and the artistic communities.

References

- 3D Coat , 2010. *3D Coat: Voxel sculpting*. Available from: <http://www.3d-coat.com/tutorial/voxel-sculpting/> [Accessed 01.10.2010].
- Abrahams D. and Grosse-Kunstleve R. W., 2009. *Building Hybrid Systems with Boost.Python*. Available from: www.boostpro.com/writing/bpl.html [Accessed 01.12.2010].
- Adzhiev V. and Beynon M., 1999. Empirical modelling of multi-agent concurrent systems with LSD-engine. In: *Empirical and Geometric Modeling, International Workshop*. University of Aizu, Japan, 13–35.
- Adzhiev V., Cartwright R., Fausett E., Ossipov A., Pasko A. and V. S., 1999. HyperFun project: a framework for collaborative multidimensional F-Rrep modeling”. In: Hughes J. and Schlick C., editors, *Implicit Surfaces 99, Eurographics/ACM SIGGRAPH Workshop*, 55–69.
- Adzhiev V., Comninos P., Kazakov M. and Pasko A., 2005. Functionally based augmented sculpting. In: *Computer Animation and Virtual Worlds*, volume 16, 25–39.
- Adzhiev V., Kartasheva E., Kunii T., Pasko A. and Schmitt B., 2002. Hybrid cellular-functional modelling of heterogeneous objects. In: *Journal of Computing and Information Science in Engineering, Transactions of the ASME*, volume 4, 312–322.
- Adzhiev V., Kazakov M., Pasko A. and Savchenko V., 2000. Hybrid system architecture for volume modeling. In: *Computers & Graphics*, volume 24, 67–78.
- Adzhiev V., Pasko A. and Sarkisov A., 1996. “HyperJazz” project: development of geometric modelling systems with inherent symbolic interactivity. In: *CSG 96 (Winchester, UK, 17-19 April 1996), Information Geometers*, 183–198.
- Alexandrov P., 1998. *Combinatorial Topology*. Dover Publications.
- Allègre R., Galin E., Chaine R. and Akkouche S., 2006. The HybridTree:

- mixing skeletal implicit surfaces, triangle meshes, and point sets in a free-form modeling system. In: *Graph. Models*, volume 68, 42–64.
- Allègre R., Barbier A., Galin E. and Akkouche S., 2004. A hybrid shape representation for free-form modelling. In: *Proceedings of the Shape Modeling International 2004*, Washington, DC, USA. IEEE Computer Society, 7–18.
- Armstrong C., Bowyer A., Cameron S., Corney J., Jared G., Martin R., Middleditch A., Sabin M. and Salmon J., 2000. *Djinn. A Geometric Interface for Solid Modelling*. Information Geometers, Winchester, UK.
- Autodesk , 2011a. *Autodesk®Maya™: 3D Animation, Visual Effects and Compositing Software*. Available from: <http://www.autodesk.com/maya> [Accessed 18.01.2011].
- Autodesk , 2011b. *Autodesk®Softimage™: Visual Effects and 3D Game Development Software*. Available from: <http://www.softimage.com>, [Accessed 18.01.2011].
- Bærentzen J. A., 2002. *Manipulation of volumetric solids with applications to sculpting*. Thesis (PhD), Informatics and Mathematical Modelling, Technical University of Denmark, DTU.
- Baran I. and Popović J., 2007. Automatic rigging and animation of 3d characters. In: *ACM Trans. Graph.*, volume 26, 72–80.
- Barbier A., Galin E. and Akkouche S., 2005. A framework for modeling, animating, and morphing textured implicit models. In: *Graphical Models*, volume 67, 166–188.
- Barr A. and Alan H., 1984. Global and local deformations of solid primitives. In: *SIGGRAPH Comput. Graph.*, volume 18, 21–30.
- Barthe L., Mora B., Dodgson N. and Sabin M., 2002. Triquadratic reconstruction for interactive modelling of potential fields. In: *SMI '02: Proceedings of the Shape Modeling International 2002 (SMI'02)*, Washington, DC, USA. IEEE Computer Society, 145–153.
- Beynon M. and Cartwright A. 1989. A definitive programming approach

- to the implementation of CAD software. In: *Intelligent CAD systems II: implementational issues*, Springer-Verlag New York, Inc., 126–145.
- Beynon W. M., 1987. *Definitive Principles for Interactive Graphics*. Technical report, Coventry, UK.
- Bezier P., 1986. *The Mathematical Basis of the UNISURF CAD System*. Butterworth-Heinemann, Newton, MA, USA.
- Biswas A., Shapiro V. and Tsukanov I., 2004. Heterogeneous material modeling with distance fields. In: *Comput. Aided Geom. Des.*, volume 21, Amsterdam, The Netherlands, The Netherlands. Elsevier Science Publishers B. V., 215–242.
- Blinn J. F., 1982. A generalization of algebraic surface drawing. In: *ACM Trans. Graph.*, volume 1, 235–256.
- Bloomenthal J., August 1997. *Introduction to Implicit Surfaces, First Edition (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann Publishers Inc.
- Bloomenthal J., 1994. An implicit surface polygonizer. In: *Graphics Gems IV*. Academic Press, 324–349.
- Bloomenthal J. and Shoemake K., 1991. Convolution surfaces. In: *SIGGRAPH Comput. Graph.*, volume 25. ACM, 251–256.
- Booch G., 2004. *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.
- Bremer P., Porumbescu S. D., Kuester F., Hamann B., Joy K. I. and liu Ma K., 2002. Virtual clay modeling using adaptive distance fields. In: *Proceedings of the 2002 International Conference on Imaging Science, Systems, and Technology (CISST)*.
- Burtnyk N. and Wein M., 1976. Interactive skeleton techniques for enhancing motion dynamics in key frame animation. In: *Commun. ACM*, volume 19, New York, NY, USA. ACM, 564–569.

- Cani-Gascuel M.-P., 1998. Layered deformable models with implicit surfaces. In: *Graphics Interface*, 201–208.
- Carlson M., Mucha P. J. and Turk G., 2004. Rigid fluid: animating the interplay between rigid bodies and fluid. In: *ACM Trans. Graph.*, volume 23, 377–384.
- Cartwright R., 1998. *Geometric Aspects of Empirical Modelling: Issues in Design and Implementation*. Thesis (PhD), Department of Computer Science, University of Warwick, UK.
- Cartwright R., Adzhiev V., Pasko A., Goto Y. and Kunii T., 2005. Web-based shape modeling with hyperfun. In: *IEEE Computer Graphics and Applications*, volume 25, 60–69.
- Catmull E., 1972. A system for computer generated movies. In: *ACM '72: Proceedings of the ACM annual conference*, New York, NY, USA. ACM, 422–431.
- Catmull E., 1978. The problems of computer-assisted animation. In: *SIGGRAPH Comput. Graph.*, volume 12, New York, NY, USA. ACM, 348–353.
- Chen M., Vucker J. and Leu A., 1999. Constructive representations of volumetric environments. In: *Volume Graphics*, M. Chen, A. Kaufman, R. Yagel (Eds.), 97–118.
- Cignoni P., Corsini M. and Ranzuglia G., April 2008. Meshlab: an open-source 3d mesh processing system. In: *ERCIM News*, number 73, 45–46.
- Clark J. H., 1976. Hierarchical geometric models for visible surface algorithms. In: *Commun. ACM*, volume 19. ACM, 547–554.
- Clavet S., Beaudoin P. and Poulin P., 2005. Particle-based viscoelastic fluid simulation. In: *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 219–228.
- Cohen-Or D., Levin D. and Solomovici A., 1996. Contour blending using warp-guided distance field interpolation. In: *VIS '96: Proceedings of the 7th conference on Visualization '96*, Los Alamitos, CA, USA. IEEE Computer Society Press, 165–172.

- Comninos P., 1986. The CGAL Animation Environment and its Application in the Entertainment Industry. In: *Proceedings of the international electronic image week, second image symposium*, 324–332.
- Comninos P. and Webster G., 1980. CGAL: Computer graphics and animation language. In: *In Eurographics Conference Proceedings*, 113–126.
- Coons S., 1967. Surfaces for computer aided design of space forms. In: *Technical Report (MIT/LCS/TR-41)*, MIT, Cambridge, Massachusetts.
- Cormen T. H., Stein C., Rivest R. L. and Leiserson C. E., 2001. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition.
- Cutler B., Dorsey J., McMillan L., Müller M. and Jagnow R., 2002. A procedural approach to authoring solid models. In: *ACM Trans. Graph.*, volume 21. ACM, 302–311.
- Desbrun M. and Gascuel M.-P., 1995. Animating soft substances with implicit surfaces. In: *In Proc. SIGGRAPH 95*. ACM SIGGRAPH, Addison Wesley, 287–290.
- Farin G., 2002. *Curves and surfaces for computer aided geometric design: a practical guide*. Morgan Kaufmann Publishers Inc.
- Fausett E., Pasko A. and Adzhiev V., 2000. Space-time and higher dimensional modeling for animation. In: *CA '00: Proceedings of the Computer Animation*, Washington, DC, USA. IEEE Computer Society, 156–165.
- Fayolle P.-A., Pasko A., Schmitt B. and Mirenkov N., 2006. Constructive heterogeneous object modeling using signed approximate real distance functions. In: *Journal of Computing and Information Science in Engineering*, volume 6, 221 – 229.
- Fayolle P.-A., Schmitt B., Goto Y. and Pasko A., 2005. Web-based constructive shape modeling using real distance functions. In: *IEICE Transactions on Information and Systems*, volume E88-D, 828–835.
- Fayolle P.-A., 2006. *Construction Of Volumetric Object Models Using Distance-Based Scalar Fields*. Thesis (PhD), University of Aizu.
- Fiume E., Tschritzis D. and Dami L., 1987. A temporal scripting language

- for object-oriented animation. In: *Proceedings of Eurographics 1987 (North-Holland), Elsevier Science*, 283–294.
- Flórez J., Sbert M., Sainz M. A. and Vehí J., 2008. Efficient ray tracing using interval analysis. In: *Proceedings of the 7th international conference on Parallel processing and applied mathematics, PPAM'07*, Berlin, Heidelberg. Springer-Verlag, 1351–1360.
- Foley J. D., van Dam A., Feiner S. K. and Hughes J. F., 1995. *Computer Graphics: Principles and Practice in C*. Pearson.
- Forstmann S., Moll S. and Ohya J., 2007. Visualization of large RLE-encoded voxel volumes. In: *Submitted to FIT 2007 (Technical Report)*, Nagoya, Japan.
- Foster N. and Fedkiw R., 2001. Practical animation of liquids. In: *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 23–30.
- Fougerolle Y. D., Gribok A., Foufou S., Truchetet F. and Abidi M. A., 2005. Boolean operations with implicit and parametric representation of primitives using r-functions. In: *IEEE Transactions on Visualization and Computer Graphics*, volume 11, Piscataway, NJ, USA. IEEE Educational Activities Department, 529–539.
- Fox M., Galbraithy C. and Wyvill B., 2001. Efficient use of the BlobTree for rendering purposes. In: *SMI '01: Proceedings of the International Conference on Shape Modeling & Applications*, Washington, DC, USA. IEEE Computer Society, 306.
- France L., Lenoir J., Angelidis A., Meseure P., Cani M.-P., Faure F. and Chailou C., 2005. A layered model of a virtual human intestine for surgery simulation. In: *Med. Images Anal.*, volume 9, 123–132.
- Friskén S. F., 1999. *Calculating the distance map for binary sampled data*. Technical Report TR99-26, December 1999, Mitsubishi Electric Research Laboratories, 201 Broadway, Cambridge, MA, USA.
- Friskén S. F., Perry R. N., Rockwood A. P. and Jones T. R., 2000. Adaptively sampled distance fields: a general representation of shape for computer

- graphics. In: *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 249–254.
- Fryazinov O. and Pasko A., 2008. Interactive ray shading of FRep objects. In: *WSCG' 2008, Communications Papers proceedings, 4-7 February, 2008, University of West Bohemia, Plzen - Bory, Czech Republic*, 145–152.
- Fryazinov O., Pasko A. and Adzhiev V., March 2011. BSP-fields: An exact representation of polygonal objects by differentiable scalar fields based on binary space partitioning. In: *Computer-Aided Design*, volume 43. Butterworth-Heinemann, 265–277.
- Fryazinov O., Pasko A. and Comninos P., 2010. Technical section: Fast reliable interrogation of procedurally defined implicit surfaces using extended revised affine arithmetic. In: *Comput. Graph.*, volume 34, Elmsford, NY, USA. Pergamon Press, Inc., 708–718.
- Galbraith C., Prusinkiewicz P. and Wyvill B., 2000. Modeling murex cabritii sea shell with a structured implicit surface. In: *CGI '00: Proceedings of the International Conference on Computer Graphics*, Washington, DC, USA. IEEE Computer Society, 55–64.
- Galin E., Leclercq A. and Akkouche S., 2000. Morphing the BlobTree. In: *Comput. Graph. Forum*, volume 19, 257–270.
- Gamma E., Helm R., Johnson R. and Vlissides J., 1995. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional.
- Gielis J., Beirinckx B. and Bastiaens E., 2003. Superquadrics with rational and irrational symmetry. In: *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, New York, NY, USA. ACM, 262–265.
- Hackathorn R. J., 1977. Anima II: a 3-d color animation system. In: *SIGGRAPH '77: Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, New York, NY, USA. ACM, 54–64.
- Heckbert P. S., November 1986. Survey of Texture Mapping. volume 6, 56–67.

- Hilton A. and Stoddart A., 1996. Marching triangles: Range image fusion for complex object modeling. In: *Image Processing*, volume 1, 381–383.
- Jones M. W., 2003. Melting Objects. volume 11, 247–254.
- Jones M. W., Bærentzen J. A. and Sramek M., 2006. 3d distance fields: A survey of techniques and applications. In: *IEEE TRANSACTIONS ON VISUALIZATION AND COMPUTER GRAPHICS*, volume 12, 581–599.
- Junior A. d. C., Figueiredo L. H. d. and Gattass M., 1999. Interval methods for ray casting implicit surfaces with affine arithmetic. In: *Proceedings of the XII Brazilian Symposium on Computer Graphics and Image Processing, SIBGRAPI '99*, Washington, DC, USA. IEEE Computer Society, 65–72.
- Junker G., 2006. *Pro OGRE 3D Programming (Pro)*. Apress, Berkely, CA, USA.
- Kadosh A., Cohen-Or D. and Yagel R., 2003. Tricubic interpolation of discrete surfaces for binary volumes. In: *IEEE Transactions on Visualization and Computer Graphics*, volume 9, Los Alamitos, CA, USA. IEEE Computer Society, 580–586.
- Kalra D. and Barr A. H., 1992. Modeling with time and events in computer animation. In: *Comput. Graph. Forum*, volume 11, 45–58.
- Kanzow C., Yamashita N. and Fukushima M., 2005. Levenberg-Marquardt methods with strong local convergence properties for solving nonlinear equations with convex constraints. In: *J. Comput. Appl. Math.*, volume 173, 321–343.
- Kartasheva E., Adzhiev V., Pasko A., Fryazinov O. and Gasilov V., 2003. Surface and volume discretization of functionally based heterogeneous objects. volume 3, 285–294.
- Kartasheva E., Adzhiev V., Comninos P., Fryazinov O. and Pasko A. 2008. An implicit complexes framework for heterogeneous objects modelling. In: *Heterogeneous Objects Modelling and Applications*, Springer-Verlag, 1–41.
- Katz S. and Tal A., 2003. Hierarchical mesh decomposition using fuzzy

- clustering and cuts. In: *ACM Transactions on Graphics (Proc. SIGGRAPH'03)*, 954–961.
- Kaufman A., Cohen D. and Yagel R., 1993. Volume graphics. In: *IEEE Computer*, volume 26, 51–64.
- Kazakov M., Pasko A. and Adzhiev V., 2001. Fast navigation through an FRep sculpture garden. In: *Shape Modeling and Applications, International Conference on*. IEEE Computer Society, 104–113.
- Knoll A., Hijazi Y., Kensler A., Schott M., Hansen C. D. and Hagen H., 2009. Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. In: *Comput. Graph. Forum*, volume 28, 26–40.
- Kobbelt L. P., Botsch M., Schwanecke U. and Seidel H.-P., 2001. Feature sensitive surface extraction from volume data. In: *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, New York, NY, USA. ACM, 57–66.
- Komma P., Fischer J., Duffner F. and Bartz D., 2007. Lossless volume data compression schemes. In: *Simulation and Visualisation 2007, Magdeburg, Germany*, 169–182.
- Kou X. Y. and Tan S. T., 2007. Heterogeneous object modeling: A review. In: *Comput. Aided Des.*, volume 39, Newton, MA, USA. Butterworth-Heinemann, 284–301.
- Kravtsov D., Fryazinov O., Adzhiev V., Pasko A. and Comninos P., 2010a. Embedded implicit stand-ins for animated meshes: a case of hybrid modelling. In: *Comput. Graph. Forum*, volume 29, 128–140.
- Kravtsov D., Fryazinov O., Adzhiev V., Pasko A. and Comninos P., 2010b. Polygonal-functional hybrids for computer animation and games. In: Engel W., editor, *GPU Pro: Advanced Rendering Techniques*, AK Peters Ltd., 87–114.
- Kravtsov D., Fryazinov O., Adzhiev V., Pasko A. and Comninos P., 2010c. Real-time controlled metamorphosis of animated meshes using polygonal-functional hybrids. In: *ACM SIGGRAPH ASIA 2010 Sketches, SA '10*, New York, NY, USA. ACM, 36:1–36:2.

- Kumar V., Burns D., Dutta D. and Hoffmann C., 1999. A framework for object modeling. volume 31, 541 – 556.
- Lasseter J., 1987. Principles of traditional animation applied to 3d computer animation. In: *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, New York, NY, USA. ACM, 35–44.
- Lavoie P., 2010. "The NURBS++ library". Available from: <http://libnurbs.sourceforge.net> [Accessed 12.12.2010].
- Lazarus F. and Verroust A., 1998. Three-dimensional metamorphosis: a survey. In: *The Visual Computer*, volume 14, 373–389.
- Leclercq A., Akkouche S. and Galin E., 2001. Mixing triangle meshes and implicit surfaces in character animation. In: *Proceedings of the Eurographic workshop on Computer animation and simulation*, New York, NY, USA. Springer-Verlag New York, Inc., 37–47.
- Liu P.-C., Wu F.-C., Ma W.-C., Liang R.-H. and Ouhyoung M., 2003. Automatic animation skeleton construction using repulsive force field. In: *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*. IEEE Computer Society, 409–413.
- LLVM Developer Group , 2010. *The LLVM Compiler Infrastructure*. Available from: <http://llvm.org> [Accessed 01.02.2010].
- Lorensen W. E. and Cline H. E., July 1987. Marching cubes: A high resolution 3d surface construction algorithm. In: *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, volume 21. ACM Press, 163–169.
- May S. F., 1998. *Encapsulated models: procedural representations for computer animation*. Thesis (PhD), The Ohio State University.
- McCormack J. and Sherstyuk A., 1998. Creating and rendering convolution surfaces. In: *Comput. Graph. Forum*, volume 17, 113–120.
- Mcnamara A., Treuille A., Popović Z. and Stam J., 2004. Fluid control using the adjoint method. In: *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, New York, NY, USA. ACM, 449–456.

- McNeel R. and Associates , 2010. "News Release: The openNURBS 4.0 toolkit released". Available from: <http://www.opennurbs.org/openNURBS4pr.pdf> [Accessed 12.10.2010].
- Moby Games™ , 2010. *Games with voxel graphics*. Available from: www.mobygames.com/game-group/games-with-voxel-graphics [Accessed 01.10.2010].
- Monaghan J., January 1988. An introduction to SPH. In: *Computer Physics Communications*, volume 48, 89–96.
- Nguyen H., 2007. *GPU GEMS 3*. Addison-Wesley Professional.
- Nielson G., 1999. Volume modelling. In: *Volume Graphics*, M. Chen, A. Kaufman, R. Yagel (Eds.), 29–48.
- NVIDIA , 2010. "NVIDIA® Compute Unified Device Architecture (CUDA™). Introduction & Overview". Available from: http://developer.download.nvidia.com/compute/cuda/docs/CUDA_Architecture_Overview.pdf [Accessed 15.04.2009].
- Oeltze S. and Preim B., 2004. Visualization of anatomic tree structures with convolution surfaces. In: *Proc. Joint IEEE/EG Symposium on Visualization. Eurographics Association, 2004*, 311–320.
- Ohtake Y., Belyaev A., Alexa M., Turk G. and Seidel H.-P., 2003. Multi-level partition of unity implicits. In: *ACM Transactions on Graphics (Proc. SIGGRAPH'03)*, volume 22, 463–470.
- Ohtake Y., Belyaev A. and Seidel H.-P., 2004. 3D scattered data approximation with adaptive compactly supported radial basis functions. In: *Proceedings of the Shape Modeling International 2004*, Washington, DC, USA. IEEE Computer Society, 31–39.
- Opalach A. and Maddock S. C., 1995. High level control of implicit surfaces for character animation. In: *Proc. 1st International Eurographics Workshop on Implicit Surfaces*, 223–232.
- Overmars M. H., 1996. Designing the computational geometry algorithms library CGAL. In: *Selected papers from the Workshop on Applied Com-*

- putational Geometry, Towards Geometric Engineering*, FCRC '96/WACG '96. Springer-Verlag, 53–58.
- Parr T., 2007. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Programmers. Pragmatic Bookshelf, first edition.
- Pasko A., Adzhiev V., Sourin A. and Savchenko V., 1995. Function representation in geometric modeling: Concepts, implementation and applications. In: *The Visual Computer*, number 11, 429–446.
- Pasko A. and Savchenko V., 1994. Blending operations for the functionally based constructive geometry. In: *Set-theoretic Solid Modelling: Techniques and Applications, CSG 94 Conference Proceedings*, 151–161.
- Pasko A. A. and Adzhiev V., 2002. Function-based shape modeling: Mathematical framework and specialized language. In: *Automated Deduction in Geometry*, 132–160.
- Pasko G., Kravtsov D. and Pasko A., 2010. Real-time space-time blending with improved user control. In: Boulic R., Chrysanthou Y. and Komura T., editors, *Lecture Notes in Computer Science: Third International Conference on Motion in Games, MIG 2010. Utrecht, the Netherlands, November 14-16, 2010*, volume 6459. Springer, 146–157.
- Pasko G., Pasko A., Ikeda M. and Kunii T., 2004a. Advanced metamorphosis based on bounded space-time blending. In: *MMM '04: Proceedings of the 10th International Multimedia Modelling Conference*. IEEE Computer Society, 211–217.
- Pasko G., Pasko A. and Kunii T., 2004b. Space-time blending. In: *Computer Animation and Virtual Worlds*, volume 15, 109–121.
- Pasko G., Nieda T., Pasko A. and Kunii T. L., 2004c. Space-time modeling and analysis. In: *SCCG '04: Proceedings of the 20th spring conference on Computer graphics*, New York, NY, USA. ACM, 13–20.
- Pasko G., Pasko A. and Kunii T., 2005. Bounded blending for function-based shape modeling. volume 25, 36–45.
- Perlin K., July 1985. An image synthesizer. In: *SIGGRAPH '85: Proceed-*

- ings of the 12th annual conference on Computer graphics and interactive techniques*, volume 19. ACM Press, 287–296.
- Popa T., Julius D. and Sheffer A., 2006. Material-aware mesh deformations. In: *SMI '06: Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*, Washington, DC, USA. IEEE Computer Society, 22.
- Prusinkiewicz P., 1986. Graphical applications of L-systems. In: *Proceedings on Graphics Interface '86/Vision Interface '86*, Toronto, Ont., Canada, Canada. Canadian Information Processing Society, 247–253.
- Python Software Foundation , 2009. *Python Programming Language*. Available from: <http://www.python.org> [Accessed 01.03.2010].
- Reeves W. T., 1983. Particle systems—a technique for modeling a class of fuzzy objects. In: *ACM Trans. Graph.*, volume 2, New York, NY, USA. ACM, 91–108.
- Reynolds C. W., 1982. Computer animation with scripts and actors. In: *Proceedings of the 9th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '82, New York, NY, USA. ACM, 289–296.
- Ricci A., 1973. A Constructive Geometry for Computer Graphics. In: *The Computer Journal*, volume 16, 157–160.
- Rossignac J. R. and O'Connor M. A. 1990. PSGC: A dimension-independent model for point-sets with internal structures and incomplete boundaries. In: *Geometric Modeling for Product Engineering*, Elsevier Science Publishers B.V. (North-Holland), Amsterdam, 145–180.
- Rossignac J., 1997. Structured topological complexes: a feature-based API for non-manifold topologies. In: *SMA '97: Proceedings of the fourth ACM symposium on Solid modeling and applications*, New York, NY, USA. ACM, 1–9.
- Savchenko V. and Pasko A., 1995. Collision detection for functionally defined deformable objects. In: *The First International Workshop on Implicit Surfaces, Eurographics Workshop*, 217–221.
- Savchenko V., Pasko A., Okunev O. and Kunii T., 1995. Function representa-

- tion of solids reconstructed from scattered surface points and contours. In: *Comput. Graph. Forum*, volume 14, 181–188.
- Schmidt R., Wyvill B. and Sousa M. C., 2005. Sketch-based modeling with the blob tree. In: *SIGGRAPH '05: ACM SIGGRAPH 2005 Sketches*, New York, NY, USA. ACM, 90.
- Schmitt B., Pasko A., Adzhiev V. and Schlick C., December 2001. Constructive texturing based on hypervolume modeling. In: *The booktitle of Visualization and Computer Animation*, volume 12, 297–310.
- Schmitt B., Pasko A. and Schlick C., 2003. Shape-driven deformations of functionally defined heterogeneous volumetric objects. In: *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. ACM, 127–134.
- Schmitt B., Pasko A. and Schlick C., 2004. Constructive sculpting of heterogeneous volumetric objects using trivariate b-splines. In: *The Visual Computer*, volume 20, 130 – 148.
- Schmitt B., 2002. *Constructive Hypervolume modelling*. Thesis (PhD), Bordeaux University I.
- Schroeder W. J., Lorensen W. E. and Linthicum S., 1994. Implicit modeling of swept surfaces and volumes. In: *VIS '94: Proceedings of the conference on Visualization '94*, Los Alamitos, CA, USA. IEEE Computer Society Press, 40–45.
- Sederberg T. W., Gao P., Wang G. and Mu H., 1993. 2-d shape blending: an intrinsic solution to the vertex path problem. In: *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. ACM, 15–18.
- Sederberg T. W. and Greenwood E., 1992. A physically based approach to 2–d shape blending. In: *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, New York, NY, USA. ACM, 25–34.
- Sederberg T. W. and Parry S. R., 1986. Free-form deformation of solid ge-

- ometric models. In: *SIGGRAPH Comput. Graph.*, volume 20, New York, NY, USA. ACM, 151–160.
- Shanat M., Fayolle P.-A., Schmitt B. and T. V., 2002. Haniwa : A case study of digital visualization of virtual heritage properties. In: *20th Eurographics UK Conference (De Montfort University, Leicester, UK)*, IEEE Computer Society, 24–32.
- Shapira M. and Rappoport A., 1995. Shape blending using the star-skeleton representation. In: *IEEE Comput. Graph. Appl.*, volume 15. IEEE Computer Society Press, 44–50.
- Shen G., Sakkalis T. and Patrikalakis N., 2001. Analysis of boundary representation model rectification. In: *SMA '01: Proceedings of the sixth ACM symposium on Solid modeling and applications*, New York, NY, USA. ACM, 149–158.
- Shi L. and Yu Y., 2005. Taming liquids for rapidly changing targets. In: *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 229–236.
- Side Effects Software , 2011. *Houdini: 3D Animation Tools*. Available from: <http://www.sidefx.com> [Accessed 18.01.2011].
- Sigg C., 2006. *Representation and Rendering of Implicit Surfaces*. Thesis (PhD), ETH Zurich, Switzerland.
- Singh K. and Parent R., April 1995. Implicit function based deformations of polyhedral objects. In: *Proc. 1st International Eurographics Workshop on Implicit Surfaces*, 113–128.
- Sourin A., Pasko A. and Savchenko V., 1996. Using real functions with application to hair modelling. In: *Computers and Graphics*, volume 20, 11–19.
- Stora D., Agliati P.-O., Cani M.-P., Neyret F. and Gascuel J.-D., 1999. Animating lava flows. In: *Proceedings of the 1999 conference on Graphics interface '99*, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc., 203–210.
- Stroustrup B., 2000. *The C++ Programming Language*. Addison-Wesley Longman Publishing Co., Inc., 3rd edition.

- Surazhsky T., Surazhsky V., Barequet G. and Tal A., 2001. Blending polygonal shapes with different topologies. In: *Computers and Graphics*, volume 25, 29–39.
- Tai C.-L., Zhang H. and Fong J. C.-K., 2004. Prototype modeling from sketched silhouettes based on convolution surfaces. In: *Comput. Graph. Forum*, volume 23, 71–84.
- Terzopoulos D., Platt J., Barr A. and Fleischer K., 1987. Elastically deformable models. In: *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, New York, NY, USA. ACM, 205–214.
- Thomas F. and Johnston O., 1995. *The Illusion of Life: Disney Animation*. Disney Editions, revised edition.
- Thürey N., Keiser R., Pauly M. and Rüdè U., 2006. Detail-preserving fluid control. In: *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*. Eurographics Association, 7–12.
- Tigges M. and Wyvill B., 1998. Texture mapping the BlobTree. In: *In Proceedings of the Third International Workshop on Implicit Surfaces*, 123–130.
- Tong M., Liu Y. and Huang T. S., 2005. Recover human pose from monocular image under weak perspective projection. In: *ICCV-HCI 2005*, 36–46.
- Uhlir K. and Skala V., February 2003. "The implicit function modeling system - comparison of C++ and C# solutions". In: *1st Int. Workshop on C# and .NET Technologies on Algorithms, Computer Graphics, Visualization, Computer Vision and Distributed Computing*, Plzen, Czech Republic. UNION Agency-Science Press, 87–92.
- Versprille K. J., 1975. *Computer-Aided Design Applications of the Rational B-Spline Approximation Form*. Thesis (PhD), Syracuse University, Syracuse, NY, USA.
- Vilbrandt C., Pasko G., Pasko A., Fayolle P.-A., Vilbrandt T., Goodwin J. R., Goodwin J. M. and Kunii T. L., 2004. Cultural heritage preservation using

- constructive shape modeling. In: *Computer Graphics Forum*, volume 23, 25–41.
- W3C , 2010. *Extensible Markup Language (XML)*. Available from: <http://www.w3.org/XML/> [Accessed 12.12.2010].
- Wirth N., 1971. The programming language Pascal. In: *Acta Inf.*, volume 1, 35–63.
- Witkin A. and Baraff D., 1997. An introduction to physically based modeling. In: *An Introduction to Physically Based Modelling, SIGGRAPH '97 Course Notes*, 1–97.
- Witkin A. and Kass M., 1988. Spacetime constraints. In: *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques*, New York, NY, USA. ACM, 159–168.
- Wyvill B., Guy A. and Galin E., 1999. Extending the csg tree. warping, blending and boolean operations in an implicit surface modeling system. In: *Computer Graphics Forum*, number 2, 149–158.
- Xiaoping Q. and Dutta D., 2003. Physics-based modeling for heterogeneous objects. In: *Journal of Mechanical Design*, volume 125. Butterworth-Heinemann, 416–427.
- Zhang Y. and Huang Y., 2000. Wavelet shape blending. In: *The Visual Computer*, volume 16, 106–115.

A A detailed description of IC entities and their components

The table presented in this section allows us to map all the theoretical concepts presented in chapter 3 to a set of more practical terms. Information in table 2 can be used as initial guidance for the implementation of the dynamic IC framework.

Entity	“Property” name	Values/Range	Description
Timespan	Span	*[REAL;REAL] set of intervals	Defines a certain interval of time.
	Scale	REAL	The scale factor of this time-span, defining the rate at which time is advanced within the span.
	Time	REAL	Local time within this time span that is updated in sync with the global time. This time is evaluated taking into account the scale factor and the start of the current interval of time.
	Global time ref	REAL	Reference to global time. This reference makes global time available for any entity present in the model at any moment.
Event	Name	Unique string	Name of the event. This name can be used by other IC entities in order to provide a reaction to a particular event.

	Timespan	Timespan	Reflects the time-span over which this event occurs. If the interval is unknown in advance, only the initial time is set to the time when the event has started.
	Parameters (State parameters)	Vector, real etc	Any parameter of a predefined type. These parameters store values providing additional information about the event.
	Priority	REAL	The priority of this event affecting the order in which its associated reactions will be performed.
Cell	Type	B-Cell, F-Cell, P-Cell, T-Cell, P-Cell etc.	Type of the cell. The type of the cell depends on the type of representation used for the definition of its shape (e.g. boundary representation (B-Cell) or function representation (F-Cell)).
	Dimensionality	0..4 or 0D..4D	The dimensionality of the cell. At the moment the maximal dimensionality is for explicitly time-dependent cells (e.g. space-time blending).

Shape	FRep API tree/ Set of BRep primitives (points, edges, triangles)/ etc	An actual shape/point-set contained in the cell. This can be an FRep tree, a mesh with all its components registered using containment relations or a composition of elements in the case of a T-cell.
Local space domain	[REAL;REAL] ^ Dimensionality	A spatial domain enclosing the current cell (or a bounding box).
Global space domain	[REAL;REAL] ^ Dimensionality	Similar to the local domain but this domain is updated according to the current transformation parameters of the cell.
Parameters (State parameters)	Vector, real, etc	Any parameter of a predefined type. These parameters provide a way to reflect and modify the current state of the cell over time. These parameters can be changed in reaction to or through the assignment of predefined animations.
Transform	Translation, rotation, scale Matrix	Transformation components or a matrix used to transform any point defined in a local coordinate system of a cell. Transforms are predefined parameters attached to each cell.

	Deformation	Mapping	A special mapping performing a modification of the point-set.
	Name	Unique string	A name can be used to identify a cell (e.g. to access a particular cell of the complex.)
	Reactions	Functions	A set of functions providing reactions to certain events.
	Priority	REAL	The priority of this cell that is used to determine the order of the cells evaluation. Cells with a higher priority are evaluated before the cells with lower a priority ³⁰ .
	Lifespan	TIMESPAN	This defines the life span of a cell (which can be infinite).
Attribute	Dimensionality	1..N	Dimensionality of the attribute. This can be an arbitrary number depending on the type of the attribute.
	Attribute mapping	Mapping function	A function performing a mapping from the modelling space to an attribute N-dimensional space.

³⁰The global order of evaluation is also affected by a set of dependency relations defined by the user.

Parameters (State parameters)	Vector, real, etc	Any parameter of a predefined type. These parameters provide a way to reflect and modify the current state of the attribute over time. These parameters can be changed in reaction to or through the assignment of predefined animations.
Transform	Translation, rotation, scale Matrix	Transformation components or a matrix used to transform any point defined in a local coordinate system of the attribute. Transforms are predefined parameters attached to each attribute.
Deformation	Mapping	A special mapping performing a modification of the point-set.
Name	Unique string	A name can be used to identify an attribute (e.g. to access a particular attribute of the complex.)
Reactions	Functions	A set of functions providing reactions to certain events.
Priority	REAL	The priority of an attribute that can be used to determine the order of the attribute evaluation. Attributes with a higher priority are evaluated before attributes with a lower priority ³¹ .

³¹The global order of evaluation is also affected by a set of dependency relations defined by the user.

	Lifespan	TIMESPAN	This defines the life span of an attribute (which can be infinite).
INSTANCE	Cells	Set of CELL entities	All the cells present in current instance of the IC. These cells can interact using parameters.
	Attributes	Set of ATTRIBUTE entities	All the attributes present in current instance of the IC. The Attributes can interact using parameters.
	Boundary/containment relations ³²	Set of cell pairs and optional names	Topological relations valid within the current instance of an IC. Each relation consists of a pair of cells. Each of the cells is present in the Cells list.
	Dependency relations	Set of pairs of cells parameters, shapes, attributes and optional names	Similar to previous relations. These relations are used to define various types of dependencies between IC entities. Dependent entities are evaluated after the entities they depend on. Bidirectional dependencies are resolved in a more sophisticated way.
	Attribute relations	Set of (Attribute; cell reference) pairs	A set of pairs associating attributes with certain cells defined in the model.

³²The number of mutual locations of the cells will grow fast due to the combinatorial nature of their relations. The user should only define instances and corresponding predicates of interest to the particular task at hand.

	Predicate	Time-dependent Boolean function	A predicate is a function of time that returns a Boolean value indicating whether the current instance of an IC is still valid (e.g. determining whether all relations specific for the current IC remain valid).
	Reactions	Functions	A set of functions providing reactions to certain events.
	Name	Unique string	The name of the instance.
	Lifespan	TIMESPAN	The life span of an instance (defined using global time).
	Parameter references	Vector, real, etc	References to parameters meaningful within this particular instance. These parameters usually reflect the state of the instance and can help us to better understand the process taking place within the life span of this instance.
	Space domain	[REAL;REAL] ³	The spatial domain used for the modelling of the current instance.
TEMPLA-TE_IC	Cells	Set of CELL entities	All the independent cells initially present in the IC. These are, for instance, B-Cells, F-Cells and others but not the cells existing during different time intervals (these cells are only present in the concrete IC instances).

Boundary/ contain- ment relations	Set of cell pairs and optional names	These relations contain rela- tions valid within the current instance of an IC. Each re- lation consists of a pair of cells. Each cell is present in the Cells list.
Dependency relations	Set of pairs of cells parameters, shapes, attributes and optional names	Similar to previous relations. These relations are used to define various types of de- pendencies between IC enti- ties. Dependent entities are evaluated after the entities they depend on. Bidirec- tional dependencies are re- solved in a more sophisti- cated way.
Attributes	Set of ATTRI- BUTE entities	All the attributes present in the current instance of the IC. The Attributes can inter- act using parameters.
Attribute relations	A set of (Attri- bute; cell refer- ence) pairs	Set of pairs associating at- tributes with certain cells de- fined in the model.
Lifespan	TIMESPAN	The life span of the overall IC. Can be thought of as the time domain used for the ac- tual modelling.

	Parameter references	Vector, real, etc	References to the parameters of the cells that have meaning within the model. These parameters usually reflect the state of the model and can help us to better understand the processes taking place in the model.
	Space domain	[REAL;REAL] ³	The maximum spatial domain used for modelling (this can also be accessed by the instances or cells).
MODEL	INSTANCES	Set of IC instances	The IC instances present in the model (only one of them can be active at any given moment of time).
	Global time	REAL	The global time currently valid in a model.
	Global space domain	[REAL;REAL] [^] Dimensionality	The global modelling domain. The model is evaluated within this domain.
	Parameter references	Set of currently active parameters	A dynamically modified set of parameters that are exposed at a particular moment of time.
	Modelling context	Arbitrary parameters	Additional parameters defining the modelling context.
	Events	Set of EVENT entities	A set of custom events used within this IC model.
	Animations	Set of ANIMATION entities	A set of predefined animations used within this IC model (see description of ANIMATION below).

Table 2: A list of dynamic IC entities and their properties.

There are supplementary methods available for the user to determine any intersections/collisions between the objects. These are needed to simplify topological queries for dynamic objects (for instance, in order to find out whether the objects are touching or intersecting each other).

Additionally, we need an animation entity (see table 3). This entity can be used to define keyframe-based animation for any parameter (i.e. it is “attached” to that parameter). Animation can be defined within an IC instance or can be applied over the entire modelling process. For instance, if we only wish to apply the animation until the moment of time when the objects collide and then we wish to switch to an alternative IC instance. As soon as another instance has been activated, the animation specific to the previous instance is no longer applied to the parameter. If a different behaviour of the animation is expected, it should be described explicitly.

Entity	“Property” name	Values/Range	Description
ANIMATION	Keyframe times	REAL	The moments of time at which the key values of the animation curve are defined.
	Values	Set of Parameters (REAL / VECTOR / ...)	The values of the parameters for every moment of time stored in Keyframe times .
	Additional values	Set of Parameters (REAL / VECTOR / ...)	The values of additional parameters for every moment of time stored in Keyframe times . These values can be used to provide additional information required for specific types of interpolation.
	Time range	[REAL;REAL]	The time range of this animation (derived from Keyframe times).
	Local time	REAL	The local/current time within the range of this animation.

	Interpolation type	Linear, various cubic curves	The method or type of function used to retrieve intermediate values over time.
	Out of range type	Constant, Linear, Mirror, Loop,...	The interpolation type for time values outside the Time range (e.g. in order to gain access to the last evaluated parameter value, loop animation, mirror a curve, etc).

Table 3: *The components of the animation entity.*

B IC model description of a “Stand-in” case study

Description of persistent cells and the first IC instance (see section 4.3.1):

```

TIMESPANS {
  // time-span used for walking out animation (initialised later)
  walkTimespan : TIMESPAN;
  // time-span used for the animation of blending params
  // (initialised later):
  blendTimespan : TIMESPAN;
}
// description of persistent cells
CELLS {
  Box {
    type = F-CELL;
    shape = frep::TREE(...);
    dim = 3D;
    domain = {...};
    // this is a static cell, no reaction provided
  }

  // used to deform Andy and drive convolution
  Skeleton {
    type = C-CELL;
    // this cell is created using union between 2 existing cells
    shape = loadSegmentsHierarchy(...);
    dim = 3D;
    domain = {...};
    parameters {
      // the actual transforms (associated with every vertex
      // in the list of segments)
      matrices : array of TRANSFORM;
      // the initial shape of the skeleton in neutral pose
      initialShape : SHAPE(shape);
    }
    REACTIONS {
      // update the shape using defined animation
      update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
      {
        // use the externally defined animation to retrieve a
        // set of up-to-date transforms (animation is played
        // depending on the external time-span)
        matrices = animations( "skeletonAnim" ).transforms(
                               walkTimespan.t );
        // use the up-to-date transforms to modify current
        // position of the skeleton
        shape = updateShape(initialShape,
                             matrices( walkTimespan.t ) );
      }
    }
  }
}
// mesh
Andy {
  type = B-CELL;
  shape = loadMesh(...);
  dim = 3D;
}

```

```

domain = {...};
parameters {
  // skeleton that can be provided to perform deformation
  transforms : array of TRANSFORM;
}
REACTIONS {
  update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
  {
    // get access to internal params of BRep model
    BREP_CELL brepCell = getBCell();
    // define skeleton using current up-to-date transform
    brepCell.setDeformation( transforms );
  }
}
}
// "Stand-in" making Andy model blendable
Liquid_Andy {
  type = F-CELL;
  shape = frep::TREE(...);
  dim = 3D;
  domain = {...};
  parameters {
    // skeleton that can be provided to define convolution
    skeleton : SHAPE;
  }
  REACTIONS {
    update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
    {
      // get access to internal params of FRep model
      FREP_CELL frepCell = getFCell();
      // submit skeleton from this cell to FRep tree
      frepCell.getFRepParam( "skeleton" ).set( "skeleton" );
    }
  }
}
// viscous liquid glued to Andy (result of blend)
Liquid_on_Andy {
  type = T-CELL;
  // this cell is created using union between 2 existing cells
  //use the internal parameter to control the amount of blending
  shape = IC::blendCells(Box, Liquid_Andy, blendDensity,...);
  dim = 3D;
  domain = {...};
  // the parameters controlling the blending
  parameters {
    // the amount of blending
    blendDensity : REAL(...);
  }
}
} // CELLS
// this is an IC in which all basic relations remain unchanged
TEMPLATE_IC1 : TEMPLATE_IC {
  // references to a set of earlier described cells
  // they exist in all instances in this example

```

```

CELLS {
    Box Skeleton Andy Liquid_Andy Liquid_on_Andy;
}
// a set of relations between the cells/parameters referenced
// in the CELLS section
RELATIONS {
    // list of containment relations
    containment {
        // can define them automatically when defining the shape
        // or explicitly
        :
    }
    // list of boundary relations
    boundary {
        // can define them automatically when defining the shape
        // or explicitly
        :
    }
    // list of dependency relations between the parameters
    dependency {
        // dependency and its priority (optional) - default
        // priority is 1.0
        Skeleton.matrices Andy.transforms;
        Skeleton.shape Liquid_Andy.skeleton;
    }
} // RELATIONS
// attaching attributes to cells
ATTRIBUTES {
    // use external mapping describing attributes:
    boxAttr = boxMapper3DTo4D;
    // volumetric definition of the liquid material
    liquidAttr = liquidMapper3DTo4D;
    :
    // establish relations between the cells
    // and attribute mappings
    RELATIONS {
        Box boxAttr;
        Liquid_on_Andy liquidAttr;
        :
    }
}
} // TEMPLATE_IC1

parameters {
    // show current blend density
    Liquid_on_Andy.blendDensity blendDensity;
    // show position of the root bone of the skeleton
    Skeleton.matrices[0].position characterCenter;
    :
}

// when Andy is still inside the box
ANDY_WITHIN_BOX {
    STATE_PARAMETERS {
        // to know how close the skeleton is to the box

```

```

        distanceToBox distance;
        :
    }
    // parameters of this instance
    parameters {
        // distance between the skeleton and the box
        // used to find out when the instance should be disabled
        distanceToBox : REAL;
    }
    // references to a set of earlier described cells
    CELLS {
        // copy all the cells from the template excluding
        // some of them
        USE TEMPLATE_IC1.CELLS;
    }
    // references to a set of earlier described cells
    RELATIONS {
        dependency {
            // copy all relations from a template IC
            USE TEMPLATE_IC1.RELATIONS.dependency;
        }
        containment {
            USE TEMPLATE_IC1.RELATIONS.containment;
        }
        boundary {
            USE TEMPLATE_IC1.RELATIONS.boundary;
        }
    }
    // attaching attributes to cells
    ATTRIBUTES {
        USE TEMPLATE_IC1.ATTRIBUTES;
    }
    // events processed by the instance
    REACTIONS {
        //
        init( REAL globalT)
        {
            // walk span starts in this instance
            walkTimespan.start();
        }
        // update distance
        update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
        {
            distanceToBox = Box.domain.distance(Skeleton.curDomain);
        }
    }
    PREDICATE {
        // predicate used to find out if the instance is still
        // valid (can be based on time or anything else within
        // the scope of the instance)
        bool evaluate( REAL globalT, REAL localT, REAL dt)
        {
            return distanceToBox < ...;
        }
    }

```

```

    }
  }
} // ANDY_WITHIN_BOX

```

Second IC instance of the model:

```

// when Andy came out of box
ANDY_OUT_OF_BOX {
  STATE_PARAMETERS {
    // to know how close the skeleton is to the box
    distanceToBox distance;
    // see the amount of blend
    Liquid_on_Andy.blendDensity amountOfBlend;
    :
  }
  // parameters of this instance
  parameters {
    // distance between the skeleton and the box
    // used to find out when the instance should be disabled
    distanceToBox : REAL;
  }
  // references to a set of earlier described cells
  CELLS {
    // copy all the cells from the template excluding
    // some of them
    USE TEMPLATE_IC1.CELLS;
  }
  REACTIONS
  {
    // provide custom reaction for the liquid
    // start eroding the shell in this instance
    Liquid_on_Andy.update( REAL globalT, REAL localT,
                          REAL lifeT, REAL dt)
    {
      // make blend coefficient proportional to the
      // distance between the skeleton and the liquid box
      blendDensity = 1 / (distanceToBox * distanceToBox);
    }
  }
}
// references to a set of earlier described cells
RELATIONS {
  dependency {
    // copy all relations from a template IC
    USE TEMPLATE_IC1.RELATIONS.dependency;
  }
  containment {
    USE TEMPLATE_IC1.RELATIONS.containment;
  }
  boundary {
    USE TEMPLATE_IC1.RELATIONS.boundary;
  }
}
// attaching attributes to cells

```

```
ATTRIBUTES {
    USE TEMPLATE_IC1.ATTRIBUTES;
}
// events processed by the instance
REACTIONS {
    // update distance
    update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
    {
        // update distance to box
        distanceToBox = Box.domain.distance(Skeleton.curDomain);
    }
}
PREDICATE {
    // predicate used to find out if the instance is still
    // valid (can be based on time or anything else within
    // the scope of the instance)
    bool evaluate( REAL globalT, REAL localT, REAL dt)
    {
        return distanceToBox < ...;
    }
}
} // ANDY_OUT_OF_BOX
```

C IC description of “Andyhausen” experiment

The IC template for this model is defined as follows:

```
// this is an IC in which all basic relations remain unchanged
TEMPLATE_IC1 : TEMPLATE_IC {
  // references to a set of earlier described cells
  // they exist in all instances in this example
  CELLS {
    Handle1 Handle2 Barrel Rope Hook Egg_contents Liquid Melting_egg;
  }
  // a set of relations between the cells/parameters referenced in
  // the CELLS section
  RELATIONS {
    // list of containment relations
    containment {
      ...
    }
    // list of boundary relations
    boundary {
      ...
    }
    // list of dependency relations between the parameters
    dependency {
      // dependency and its priority (optional) - default
      // priority is 1.0
      Handle1.alpha Handle2.alpha;
      Handle2.alpha Handle1.alpha 0.5;
      Handle1.alpha Barrel.alpha;
      Barrel.distTravelled Rope.inactiveLength;
      Rope.endPoint Hook.transform.translation;
    }
  } // RELATIONS
  // attaching attributes to cells
  ATTRIBUTES {
    // use external mapping describing attributes:
    liquidAttr = liquidMapper3DTo4D;
    // contains white and yolk distribution:
    eggContentsAttr = eggMapper3DTo4D;
    ...
    // establish relations between the cells and
    // the attribute mappings
    RELATIONS {
      Liquid liquidAttr;
      Egg_contents eggContentsAttr;
      ...
    }
  }
} // TEMPLATE_IC1
```

The first IC instance is defined in the following way :

```
// first instance of the IC
EGG_GOING_DOWN {
  STATE_PARAMETERS {
    // to know how close the egg has moved to the liquid
    distanceToLiquid liquidToEggDistance;
```

```

    :
  }
  // the parameters of this instance
  parameters {
    // period of time it takes to bring egg to the liquid
    descentDuration : REAL(...);
    // distance between the egg and the surface
    distanceToLiquid : REAL(...);
  }
  // references to a set of earlier described cells
  CELLS {
    //copy all the cells from the template except the melting egg
    USE TEMPLATE_IC1.CELLS \ { Melting_egg };
    Eggshell;
    :
    // custom REACTIONS of the referenced cells
    // (i.e. procedures called)
    REACTIONS
    {
      // this one uses all default REACTIONS of the cells
    }
    :
  }
  // references to a set of earlier described cells
  RELATIONS {
    dependency {
      // copy all relations from a template IC
      USE TEMPLATE_IC1.RELATIONS.dependency
      Hook.transform Egg_shell.transform HIERARCHICAL
      Hook_egg_dependency;
      Egg_shell.transform Egg_contents.transform HIERARCHICAL
      Egg_contents.dependency;
    }
    containment {
      USE TEMPLATE_IC1.RELATIONS.containment;
      Eggshell Eggcontents;
    }
    boundary {
      USE TEMPLATE_IC.RELATIONS.boundary;
      Hook Eggshell; // eggshell attached to a hook
    }
  }
  // attaching attributes to cells
  ATTRIBUTES {
    TEMPLATE_IC1.ATTRIBUTES;
    // use external mapping describing attributes:
    liquidAttr = liquidMapper3DTo4D;
    eggshellAttr = eggShellMapper3DTo4D;
    :
    // attach these attributes to some cells now
    RELATIONS {
      Liquid.attributes = liquidAttr;
      Eggshell.attributes = eggshellAttr;
    }
  }

```

```

        :
    }
}
// events processed by the instance
REACTIONS {
    // initialisation, destruction and update
    update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
    {
        // the angular velocity "omega" can be varied as well
        Handle.alpha += Handle.omega * dt;
        // evaluate distance to the liquid for the egg
        distanceToLiquid = Liquid.domain.distance(
            Eggshell.curDomain);
    }
}
PREDICATE {
    // predicate used to find out if the instance is still valid
    // (can be based on time or anything else within the scope
    // of the instance)
    bool evaluate( REAL globalT, REAL localT, REAL dt)
    {
        // instance becomes invalid when the eggshell touches
        // the liquid
        return !Liquid.domain.isIntersection(Eggshell.curDomain);
    }
}
:
}
} // EGG_GOING_DOWN

```

After describing the first IC instance we introduce the second instance (the egg touching the surface, the shell starting to crack/erode, while the contents of the egg starts blending with the liquid):

```

// second instance of the IC
EGG_ERODING {
    STATE_PARAMETERS {
        // show the state of the eggshell, while it's active
        Eggshell.damageState eggtermination;
        // blend rate to know how strongly the contents have
        // been blended
        Melting_egg.blendRate meltRate;
        :
    }
    // the parameters of this instance
    parameters {
        // period of time it takes to bring egg to the liquid
        descentDuration : REAL(...);
    }
    // references to a set of earlier described cells
    CELLS {
        // copy all the cells from the template except

```

```

// the melting egg
USE TEMPLATE_IC1.CELLS \ { };
Eggshell;

:
// custom REACTIONS of the referenced cells
// (i.e. procedures called)
REACTIONS
{
  // start eroding the shell in this instance
  Eggshell.update( REAL globalT, REAL localT,
                  REAL lifeT, REAL dt)
  {
    // get the F-Cell representing the egg
    FREP_CELL frepCell = getFCell();
    // evaluate how much the egg has been damaged:
    damageState = lifeT * damageRate;
    // use the damage state to modify the noise
    // subtracted from the surface:
    // subtracted from the surface:
    frepCell.getFRepParam("noiseParam").set(damageState);
  }
}
:
}

// references to a set of earlier described cells
RELATIONS {
  dependency {
    // copy all relations from a template IC
    USE TEMPLATE_IC1.RELATIONS.dependency
    Hook.transform Egg_shell.transform HIERARCHICAL
                                     Hook_egg_dependency;
    Egg_shell.transform Egg_contents.transform HIERARCHICAL
                                     Egg_contents_dependency;
  }
  containment {
    USE TEMPLATE_IC1.RELATIONS.containment;
    Eggshell Eggcontents;
  }
  boundary {
    USE TEMPLATE_IC.RELATIONS.boundary;
    Hook Eggshell; // eggshell attached to a hook
  }
}

// attaching attributes to cells
ATTRIBUTES {
  TEMPLATE_IC1.ATTRIBUTES;
  // use an external mapping describing attributes:
  liquidAttr = liquidMapper3DTo4D;
  eggshellAttr = eggShellMapper3DTo4D;
  :
  // establish the relations between the cells and
  // attribute mappings
  RELATIONS {
    Liquid liquidAttr;

```

```

        Eggshell eggshellAttr;
        :
    }
}
// events processed by the instance
REACTIONS {
    // initialisation, destruction and update
    update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
    {
        // the angular velocity "omega" can be varied as well
        Handle.alpha += Handle.omega * dt;
    }
}
PREDICATE {
    // predicate used to find out if the instance is still
    // valid (can be based on time or anything else within
    // the scope of the instance)
    bool evaluate( REAL globalT, REAL localT, REAL dt)
    {
        // instance becomes invalid when the eggshell completely
        // sinks in the liquid
        return ! Liquid.domain.isInside(Eggshell.curDomain);
    }
}
:
}
} // EGG_ERODING

```

The third IC instance contains a circular dependency caused by the dependency relation between the cell **Andy** and **handle2**. The cell “Eggshell” does not exist any longer and there is no dependency between the cell **egg liquid** and the **hook**. We add a dependency between the **hook** and the skeleton instead:

```

// when Andy came out of the egg
ANDY_OUT {
    STATE_PARAMETERS {
        // how high has andy pulled herself:
        Andy.curDomain.Y andyHeight;
        :
    }
    // references to a set of earlier described cells
    CELLS {
        //copy all the cells from the template
        USE TEMPLATE_IC1.CELLS
        // used to deform Andy and to drive the convolution
        Skeleton {
            type = C-CELL;
            //this cell is created using union between 2 existing cells
            shape = loadSegmentsHierarchy(...);
            dim = 3D;
        }
    }
}

```

```

domain = {...};
REACTIONS {
    // update the shape using defined animation
    update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
    {
        // use externally defined animation to retrieve
        // current position of the skeleton
        shape=animations( "skeletonAnim" ).params(localT);
    }
}
// mesh
Andy {
    type = B-CELL;
    shape = loadMesh(...);
    dim = 3D;
    domain = {...};
    parameters {
        // skeleton that can be provided to perform deformation
        skeleton : array of TRANSFORM;
    }
    REACTIONS {
        update(REAL globalT, REAL localT, REAL lifeT,REAL dt)
        {
            // get access to internal params of the BRep model
            BREP_CELL brepCell = getBCell();
            // define skeleton using current
            // up-to-date transform
            brepCell.setDeformation( "skeleton" );
        }
    }
}
// the "Stand-in" making Andy model blendable
Liquid_Andy {
    type = F-CELL;
    shape = frep::TREE(...);
    dim = 3D;
    domain = {...};
    parameters {
        // the skeleton that can be provided
        // to define convolution
        skeleton : array of VECTOR3;
    }
    REACTIONS {
        update( REAL globalT, REAL localT,
            REAL lifeT, REAL dt)
        {
            //get access to the internal params of the FRep model
            FREP_CELL frepCell = getFCell();
            // submit skeleton from this cell to FRep tree
            frepCell.getFRepParam("skeleton").set("skeleton");
        }
    }
}
// viscous liquid glued to Andy (result of the blend)

```

```

Liquid_on_Andy {
  type = T-CELL;
  // this cell is created using union
  // between 2 existing cells
  shape = IC::blendCells(Melting.egg, Liquid_on_Andy, ...);
  dim = 3D;
  domain = {...};
  // it doesn't have any behaviour as it is implicitly
  //dependent on two other cells
}

// custom REACTIONS of the referenced cells
REACTIONS
{
  // this one uses all the default REACTIONS of the cells
}

}

// references to a set of earlier described cells
RELATIONS {
  dependency {
    // copy all relations from a template IC
    // (but remove egg - hook dependency)
    USE TEMPLATE_IC1.RELATIONS.dependency \
      { Hook_egg_dependency Egg_contents_dependency};
    // instead attach skeleton to the rope
    Hook.transform Skeleton.transform HIERARCHICAL;
    // the skeletons driving "both Andys"
    Skeleton.shape Andy.skeleton;
    Skeleton.shape Liquid_Andy.skeleton;
    // the dependency between the hand on the skeleton
    // and the handle:
    Andy.skeleton[ "hand" ].rotationZ Handle2.alpha;
  }
  containment {
    USE TEMPLATE_IC1.RELATIONS.containment
    // some relations for BReps can be derived automatically
    :
  }
  boundary {
    USE TEMPLATE_IC1.RELATIONS.boundary
    Hook Skeleton
    // some relations for BReps can be derived automatically
    :
  }
}

// attaching attributes to cells
ATTRIBUTES {
  // use an external mapping describing the attributes:
  liquidAttr = liquidMapper3DTo4D;
  // contains the white and yolk distribution:
  eggContentsAttr = eggMapper3DTo4D;
  // establish the relations between the cells
  // and the attribute mappings
  RELATIONS {
    Liquid_Andy liquidAttr;
  }
}

```

```
        Liquid_on_Andy liquidAttr;
        eggContents eggContentsAttr;
        :
    }
}
// events processed by the instance
REACTIONS {
    // initialisation, destruction and update
    update( REAL globalT, REAL localT, REAL lifeT, REAL dt)
    {
        // use a predefined animation to drive the skeleton
        // which will result in many other actions
        Skeleton.shape = animations( "skeletonAnim" ).params(
            localT);
    }
}
PREDICATE {
    // predicate used to find out if the instance is still valid
    // (can be based on time or anything else within the scope
    // of the instance)
    bool evaluate( REAL globalT, REAL localT, REAL dt)
    {
        // use the duration of the animation to see how long
        // this will be happening for
        return localT < animations( "skeletonAnim" ).duration;
    }
}
} // ANDY_OUT
```

D List of publications

Kravtsov D., Fryazinov O., Adzhiev V., Pasko A. and Comninos P., 2010a. Embedded implicit stand-ins for animated meshes: a case of hybrid modelling. In *Comput. Graph. Forum*, 29(1), 128–140.

Kravtsov D., Fryazinov O., Adzhiev V., Pasko A. and Comninos P., 2010b. Polygonal-Functional Hybrids for Computer Animation and Games. In *Engel W., editor, GPU Pro: Advanced Rendering Techniques*, AK Peters Ltd, 87–114

Pasko G., Kravtsov D., Pasko A., 2010c. Real-Time Space-Time Blending with Improved User Control, *The 3-d International Conference on Motion in Games MIG10, Zeist, the Netherlands, November 1316, 2010, Lecture Notes in Computer Science, Eds.: Boulic, Y. Chrysantou, and T. Komura*, Springer, Heidelberg, LNCS 6459, 146–157.

Kravtsov D., Fryazinov O., Adzhiev V., Pasko A. and Comninos P., 2010d. Real-time controlled metamorphosis of animated meshes using polygonal-functional hybrids. In *ACM SIGGRAPH ASIA 2010 Sketches, Seoul, Korea. SA'10*, New York, NY, USA. ACM, pages 36:1–36:2.

Pasko G., Kravtsov D., Pasko A., 2010e. Real-Time Controlled Space-Time Blending. In *ACM SIGGRAPH ASIA 2010 Sketches, Seoul, Korea. SA'10*, New York, NY, USA. ACM, 38:1–38:2.

Kravtsov D., Fryazinov O., Adzhiev V., Pasko A. and Comninos P., 2010f. Controlled metamorphosis of animated meshes using polygonal-functional hybrids. In *Poster Proceedings. SIGGRAPH 2010, Los Angeles, California, USA*, New York, NY, USA. ACM.

Kravtsov D., Fryazinov O., Adzhiev V., Pasko A. and Comninos P., 2009. Polygonal-functional hybrids for computer animation and games. In *Poster Proceedings. ACM SIGGRAPH 2009, New Orleans, Louisiana, USA*. New York, NY, USA. ACM.