No More Reinventing the Virtual Wheel: Middleware for Use in Computer Games and Interactive Computer Graphics Education

Eike Falk Anderson
Interactive Worlds ARG
Coventry University, UK

Christopher E. Peters
Interactive Worlds ARG
Coventry University, UK

Abstract

The creation of application frameworks for teaching computer graphics has always been a time-consuming task, alleviated by the use of graphics API's that allow students more scope in investigating and creating graphics programs. A boom in Open Source software and the increasing growth of the game middleware industry has created a huge number of alternatives to choose from. Despite the adoption of such API's for teaching and experimentation, centralised sources of information regarding components are not commonplace; the process for gathering information about components may be cumbersome and is often left to chance. Here, we provide details of accessible middleware of relevance to the teaching of computer games and graphics curricula. Additionally, we describe concerns and considerations when introducing the use of middleware into a curriculum.

1 Introduction

In computer graphics education, one of the main activities that students embark on is implementing small graphics programs that allow them to experiment with newly acquired knowledge of algorithms and techniques. Similarly, games courses usually involve "small projects and assignments to exercise the concepts covered in the reading and lectures" [Jon00]. To enable the students to complete these assignments, educators often provide them with an application framework that allows them to concentrate on the task at hand, an example of which is the SAGE game engine provided to game programming students at the University of North Texas [PNS*07].

While the development of such application frameworks is a time consuming process, it is now much simpler than it was twenty years ago. In terms of graphics application development, "the most significant aspect of software evolution has been the emergence and acceptance of standardised graphics software API's (Application Programmer Interfaces)" [HS00], such as OpenGL. Computer graphics courses, from introductory to advanced level, now usually employ a top-down approach to computer graphics [ACSS06], using the graphics API's to allow students to create graphics applications. While graphics API's simplify the creation of applications by removing the burden of having to implement all elements of the graphics pipeline before images can be created, they often provide merely the relatively low-level rendering services. However, there exist a large number of libraries and API's providing the required functionality that can be used: thus far, no centralised source of information about the use of middleware for computer games and graphics education, and in particular, of the implications and pitfalls of middleware component usage in courses, has been evident in the literature. Although often assumed to be common knowledge, recommendations regarding available systems that educators can leverage in their classrooms often take the form of forum postings on the internet. Well-advertised middleware components targeted for commercial use in the games industry may be high-profile, but unsuitable for the contemporary classroom due to expense, while high-quality, Open Source and/or free alternatives may exist, but frequently go unnoticed: in our experience, useful libraries are sometimes found only by coincidence.

To rectify this, in Section 2 we introduce and rationalise the use of middleware for computer graphics education, enumerating in Section 3 a list of Open Source and easily accessible middleware that we have found to be particularly useful in our experience for games and graphics education. Additionally, in Section 4, we describe some of the concerns and considerations that should be accounted for when introducing the use of middleware into a curriculum.

2 Application Development in Games and Graphics Education

The development of small applications that implement recently learned concepts lies at the core of every computer science programming course and thus also occurs in games and graphics courses. There is considerable overlap between computer games and computer graphics education, with a high percentage of interactive computer graphics topics also featuring in technical games courses [SSR07]. Consequently the types of assignments and exercises that make up the students' activities in these courses are very similar, usually resulting in the creation of small interactive graphics applications that demonstrate a specific graphics or games technique.

2.1 Educational Rationale

Cunningham stresses that it is important to develop "a good set of projects that are meaningful to the student" [Cun99], i.e. apart from being relevant, assignments and exercises also need to *appear* relevant to the students. This increases the credibility of the course in the eyes of the students and helps maintain their interest in the subject. Furthermore the students should be able to exercise what they have learned and to complete their assignments "without requiring them to become masters of a particular sub-discipline" [HS00]. This is especially the case when graphics programming is not the primary educational goal of application development activities [Cun08]. Students from other disciplines, for example, who simply want to use graphics for visualisation of information from their chosen domain, may not have the low-level knowledge to implement their applications' graphics from the bottom-up.

To enable students to create their applications, educators need to provide them with a suitable program infrastructure. Existing application frameworks that are ready to use may not have the features required for the tasks the students have to perform, or may even complicate the students' efforts, a problem that can be averted by creating proprietary application frameworks [DW07]. However, rather than using such a proprietary framework, the development of which often takes a long time, it is possible to construct the applications using existing middleware solutions. The use of middleware is an approach that can be used in situations where the students are supposed to learn graphics principles by implementing specific op-

erations. It is also useful for assignments that require the students to apply more advanced algorithms and techniques.

A common problem, for example, is the multitude of different file formats that could be used for the provision of content for students' applications. Here, middleware API's are of use, for example those described in Section 3.1.3. They can take the shape of small libraries that address a small and specific problem, up to complex application frameworks as is the case with many computer game engines. Apart from their obvious use in game development, it is possible to use computer game engines as the renderer in graphics programming assignments [WKS08].

The middleware systems used usually provide API-level access to the libraries that implement the required functionality, which is sufficient for most student projects. However, students can benefit from the additional information that can be gained from systems that also provide source-level access. In this respect, free software that includes a source-level distribution can provide a viable alternative.

The middleware presented in this paper was selected based on free availability and portability to provide for the widest possible audience. It is by no means intended to be exhaustive or complete, but includes those systems that we have found useful. There exist many platform and/or vendor dependent API's that are useful too. However, these rely on the availability of specific hardware, programming languages or operating systems, such as DirectX, the XNA framework or various graphics-card specific libraries. Similarly there are many commercial API's that are priced above the practical budgets available to most educational institutions. These are not covered in this paper. A downside to reliance on middleware is the danger that the distribution of libraries may be discontinued or the projects, if developed as Open Source, may be orphaned. We have aimed to only include those systems that are either stable, e.g. GLUT (see Section 3.1.1), or actively maintained, and accessible for academic and educational use.

3 Middleware for Games and Graphics Applications

There are far too many middleware systems to mention here, many providing basic solutions for almost any conceivable problem. Some examples include the popular Boost C++ libraries (*http://www.boost.org*) that complement the C++ standard, maths utility libraries, such as the Foundation Library by Geometric Tools (*http://www.geometrictools.com*) or the CAL3D Character Animation Library (*http://home.gna.org/cal3d/*).

3.1 Graphics Middleware

3.1.1 Rendering Context

The creation of a suitable rendering context with framebuffer within the used windowing system is a fundamental operation that is required by every interactive computer graphics application. As this is a fairly low-level operation that needs to be close to the used hardware platform and operating system, it can be a very complex task, which is greatly simplified by a range of API's.

- GLUT
  (*http://www.opengl.org/resources/libraries/*)
  The freely available OpenGL Utility Toolkit GLUT is probably the most frequently used middleware for OpenGL graphics programming used in education. It provides basic windowing functionality for defining a window hierarchy with an OpenGL rendering context and additionally includes functions for the display of a number of wireframe and solid 3D objects, e.g. cube, sphere, torus, and even the famous Utah Teapot. These may be useful for demonstrating various techniques in the absence of an object loader. GLUT also provides a means for interpreting user input from mouse, keyboard and other devices and for creating a simple pop-up menu user interface. The official GLUT distribution, which is free and distributed as source code, is not under an Open Source licence. However, there are a number of Open Source alternatives that implement the GLUT API specification, such as freeglut (*http://freeglut.sourceforge.net*). GLUT's functionality is limited by its narrow scope - the platform independent, simple and easy to use provision of an OpenGL rendering context. Therefore it may not be the best choice for more complex applications.

- GLFW (*http://www.glfw.org*)
  GLFW is a modern Open Source alternative to GLUT that can create an OpenGL rendering context with minimal effort of two instructions. The API provides a range of alternative input modes for the keyboard (with Unicode support), mouse and joysticks. The library, usable with multi-threaded applications, has built in support for OpenGL extensions allowing the native use of newer OpenGL versions.

- SDL (*http://www.libsdl.org*)
  The Simple DirectMedia Layer [LH01] was conceived as a platform-independent Open Source alternative to Microsoft's DirectX API for multimedia application development. The basic functionality of the system includes window management, 2D video supporting blit operations, event-driven user input for keyboard, mouse and joystick, audio file playback, audio CD playback, support for multithreaded programming and a multimedia timer. SDL also provides functions for the seamless integration of OpenGL by providing an OpenGL rendering context in its video interface. Additional utility libraries allow the rendering of TrueType fonts, networking, loading of a wide range of image file formats and multichannel audio with support for different audio and music file formats.

3.1.2 Graphical User Interfaces (GUIs)

Native GUIs

One of the simplest methods for adding a GUI to an application is to use the GUI components provided by the windowing system used on the host-platform. This can be done using a wide variety of GUI systems, which often include a WYSIWYG, or What You See Is What You Get, editor to simplify the GUI design process.

- GTK+ (*http://www.gtk.org*)
  The GTK+ toolkit, originally created to facilitate the development of the GNU Image Manipulation program, GIMP, is an object oriented and platform independent widget toolkit for the creation of complex GUIs for modern desktop-based windowing systems. A separate GUI editor, called GLADE *http://glade.gnome.org*, for rapid application development (RAD) is also available for the GTK+ system. Although GTK+ itself does not offer functionality for creating a rendering context for 3D graphics, there is an OpenGL GTK+ extension provided by a third party which is available for most of the GTK+ supported platforms. See *http://gtkglext.sourceforge.net*.

- wxWidgets (*http://www.wxwidgets.org*)
  wxWidgets is an Open Source C++ library that is free for commercial and non-commercial use. It provides applications with a native look and feel by using the platforms own native API, although this can result in differing behaviour between systems and requires an awareness of platform specific bugs. It features a large number of utility classes, relating to date/time functions, networking, container objects and more.

- FLTK (*http://www.fltk.org*)
  The Fast Light Toolkit focuses on providing a lightweight alternative to other toolkits and is small enough to be statically linked. This also means that it does not provide utility functions. FLTK includes a UI builder called FLUID and supports OpenGL by emulating GLUT.

- QT (*http://qt.nokia.com*)
  QT is a cross-platform UI framework that draws its own widgets on each platform rather than using native ports. It contains multiple IDE's such as QtDesigner, QtCreator and QDevelop and also integrates with popular external IDE's, such as Microsoft Visual Studio. It features a large number of utility classes, relating to date/time functions, networking, container objects and OpenGL functionality.

In-Application GUIs

A different approach for adding GUIs to a graphics application is to overlay the GUI onto the application's framebuffer, for which there are several suitable GUI API's.

- CEGUI (*http://www.cegui.org.uk*)
  Crazy Eddie's GUI system is a C++ library targeting the creation of GUI sub-systems for games, with native support for Direct X, OpenGL and the OGRE and Irrlicht engines. CEGUI requires keyboard, mouse and other inputs to be provided for it, thus requiring the use of an external input library.

- GiGi (*http://gigi.sourceforge.net*)
  The platform independent and extensible GUI API GiGi for OpenGL based applications includes a range of common GUI controls and standard dialogs, supporting the rendering of formatted text and optional rendering of a mouse cursor. The system provides bindings to the SDL multimedia library and the OGRE game graphics engine, in addition to a rudimentary GUI editor.

3.1.3 Image Loading

Texture information for 3D objects in games and other graphics applications are usually loaded from files storing 2D images. There exist a multitude of different file formats, which can be quite complex if the image information is stored in compressed form. Image loader API's, most of which also offer some form of image processing functionality, greatly simplify the use of images.

- DevIL (*http://openil.sourceforge.net*)
  The Open Source Developer's Image Library provides image loading capabilities for a wide range of commonly used image file formats, as well as a number of computer games specific image and texture formats. In addition to image loading and saving, the library also provides a number of functions for image processing/filtering. The library integrates smoothly with a number of rendering API's including OpenGL and DirectX, but can also be used independent of these. The library's API is based on GLUT, making it particularly easy to use for those familiar with GLUT.

- FreeImage (*http://freeimage.sourceforge.net*)
  FreeImage is an extensible Open Source library that can load images in a large number of popular image file formats from different locations, including compressed files and the internet, as well as save image files. The API provides easy access to all types of image data, allowing seamless integration into applications that require access to images. For this, the library provides several colour conversion, colour manipulation and image processing functions.

- ImageMagick (*http://www.imagemagick.org*)
  The ImageMagick suite for image creation and manipulation can handle over a hundred different 2D image file formats. ImageMagick can be used for standalone image processing, including several command-line interface tools, but also includes API's that expose its image loading, writing and powerful image processing functionality to other applications. The Open Source system implements useful image manipulation operations such as transformations, multiple-image compositing and a wide range of graphical special effects.

3.1.4 3D Model Loader

The loading of 3D scenes and models is a common task for graphics applications. It greatly enriches the virtual environments created by the applications. Most available model loaders can only process a single or a few select file types. Others may be bound to a specific system infrastructure, as is the base for libg3D (*http://automagically.de/g3dviewer/*) which requires GTK+, or the model loading utility routines of the OpenSceneGraph (OSG) system requiring OSG to be used.

- Assimp (*http://assimp.sourceforge.net*)
  The Open Asset Import Library is the only platform independent Open Source solution, that we are aware of, for accessing different types of 3D model and animation data in a neutral format. The library loads a wide range of 3D mesh and animation formats that are commonly used in 3D graphics and games, including a number of games specific file formats. These are presented to users in the form of easy-to-use data structures accessible through the system's API.

3.1.5 High-level Graphics and Visualisation

If the aim of an application is focussed on context rather than the underlying technology, then it is useful to use a rendering system that provides a higher level of abstraction than the commonly used low-level graphics API's. The available systems usually integrate operations for much of the peripheral tasks of rendering systems, such as the loading of 3D models and textures, greatly simplifying the construction of complex 3D scenes.

Scene Graphs

The systems that have traditionally provided this type of functionality to CG applications have been scene graph libraries. Bouvier notes that "the educational power of the scene graph lies in the ease with which a complex scene can be represented" [Bou02], as the intrinsics of the underlying technology are hidden from the application developer, who can then focus on the development of the virtual environment.

- OpenSceneGraph (*http://www.openscenegraph.org*)
  The OpenGL based OpenSceneGraph toolkit provides high-level functionality not found in low-level graphics API's, such as complex scene management, including level of detail and large-scale terrain handling, particle effects and multithreading support. OSG "also offers a variety of utility classes like GUI support, camera manipulators, picking functionality and loaders for many common data formats" [KRW*03].

- Visualization Library
  (*http://www.visualizationlibrary.com*)
  Utilising the features offered by recent OpenGL versions, the lightweight, portable Visualization Library toolkit employs data structure separation and specialisation, which is uncommon for this type of library, but offers performance benefits. The API supports the loading of common 3D model and 2D image file formats, the use of different scene management

methods and has bindings to some of the most commonly used GUI systems. The library provides access to the underlying low-level graphics, making the system highly customisable.

Figure 1: Student project using the OGRE rendering engine by Szymon Marciniewicz.

Game Rendering Engines

For high-performance real-time grahics applications, computer game rendering engines can be used, as they often integrate many of the technologies that are required, i.e. they "efficiently use rendering pipelines, special data structures and speed-up techniques" [FK04] that constitute advanced graphics techniques, providing "superior platforms for rendering multiple views and coordinating real and simulated scenes" [LJ02]. Furthermore, many modern game rendering systems also include content creation pipelines that greatly simplify the development of rich virtual environments.

- OGRE (*http://www.ogre3d.org*)
  The Object Oriented Graphics Rendering Engine, OGRE (Figure 1), is a highly-modular 3D multi-platform graphics engine, with OpenGL and Direct3D support through abstracted class libraries. It features a scene graph based engine with support for a variety of managers, such as octree, BSP, paging landscapes and portal based managers. It boasts a wide range of modern features, supporting progressive level of details for terrains, an animation system that supports hardware weighted multiple bone skinning, texture and model loading libraries and Cg, HLSL and GLSL custom shader support. OGRE explicitly supports the OIS (a simple object-oriented input library - *http://sourceforge.net/projects/wgois/*), SDL and CEGUI libraries.

- Horde3D (*http://www.horde3d.org*)
  Horde3D is a modern OpenGL based 3D graphics engine, distributed under an Open Source licence, which supports some of the most recent real-time rendering techniques, including programmable shaders using a sophisticated materials system, a resource and scene management system with support for multiple levels of detail, and postprocessing effects. The engine also includes subsystems for character animation and particles. The API provides hooks for easy integration with other middleware.

3.2 Games Specific Middleware

3.2.1 Physics

Physics has been receiving increased scrutiny by the game industry for providing more credible gaming experiences, leading to the development of numerous middleware choices. Many of these are available for educational use, offering environment and object interaction capabilities in students' creations and also sandboxes for physics experimentation. Middleware physics solutions generally offer collision detection and response, dynamics and constraint solving capabilities as standard. They may additionally provide modelling and animation package plug-ins and specialised physics components, to simulate vehicle physics for example.

- Open Dynamics Engine (*http://www.ode.org*)
  ODE is an Open Source library with a platform independent C/C++ API for simulating rigid body dynamics. In addition to other capabilities, ODE boasts simulation of advanced joint types.

- Newton Game Dynamics
  (*http://newtondynamics.com*)
  The Newton Game Dynamics physics engine seeks to provide accuracy in simulation, with a deterministic solver that is not based on LCP or iterative methods. A new version of Newton, currently under development, is intended to provide improved simulation speed and utilise GPU's and multi-core CPU's.

- Bullet (*http://www.bulletphysics.com*)
  Bullet is a collision detection and rigid-body dynamics library, featuring COLLADA physics support and discrete and continuous collision detection.

- Havok (*http://www.havok.com*)
  Havok Physics, rated in one study as the third most used middleware in all commercial games development [DeL09], is available as a free SDK for non-commercial educational and academic use as part of an Intel-sponsored download that also includes the animation SDK. It provides an excellent opportunity to expose students to a high-quality, commercial solution.

3.2.2 Sound

While rarely used in pure graphics applications, computer games and most multimedia applications require either sound or music, which is loaded for playback during application runtime.

- FMOD (*http://www.fmod.org*)
  The FMOD audio engine provides multimedia applications with audio playback from a wide range of file formats and audio CDs. Furthermore, the library allows sound recording, provides various sound effects to be applied to audio samples and also supports 3D positioning of sound sources for surround sound. The simplicity of the API and the availability of the system on most platforms make FMOD the second most used middleware in commercial games development [DeL09] and ideal for computer games education [SSR07].

- BASS (*http://www.un4seen.com/bass.html*)
  The BASS audio library, mentioned by Sung et al. [SSR07], provides routines for loading audio files from several common sound and music file formats, for recording sound samples and for audio playback, including support for 3D surround sound. The small core library supports a large number of add-ons that provide additional functionality, such as processing of additional file formats and audio CD playback.

- OpenAL
  (*http://connect.creativelabs.com/openal/*)
  Using an API modelled after the OpenGL graphics API, the OpenAL library provides audio playback functionality based on 3D sound positioning. Like OpenGL, there is no integrated support for the loading of assets, which need to be provided by external audio file loaders. However, some limited support for audio file loading is provided by the separate ALUT OpenAL Utility Toolkit library, modelled after the GLUT API.

3.2.3 Scripting

There is an observable trend towards data-driven applications in graphics, multimedia and computer games. The embedding of a scripting system into applications and the exposure of application functionality to the scripting system is a common approach to obtain data-driven applications.

- TinyXML
  (*http://www.grinninglizard.com/tinyxml/*)
  One of the most frequent uses of scripts is in the form of initialisation scripts that set application parameters [And08], for which the eXtensible Markup Language, XML, is ideal. The TinyXML library provides an XML document parser that is easily embeddable into applications, which allows data to be accessed through a Document Object Model, DOM, for ease of use.

- Lua (*http://www.lua.org*)
  The scripting language Lua is one of the most popular choices for integrating scripting with computer games. It is a generic programming language that was originally designed to be used to extend programs by adding various scriptable features, the reason why the creators of Lua have dubbed it an "extensible extension language" [IdFC96]. Lua has a C API, making it easy to embed in C/C++ based applications. It is easy to learn, making it ideal for game development environments in which non-programmers may be required to write some scripts [Har05]. The language features provided by Lua can simplify the creation of solutions to various problems in the development of multimedia and graphics applications, such as computer games. Lua has been used extensively in computer games development, being embedded in a large number of best-selling computer games.

- Python (*http://www.python.org*)
  Python is a powerful and feature-rich scripting language that also allows some object orientation. It is a general purpose language that can be used as a standalone command interpreter. It has also been used as an embedded scripting environment for various computer graphics applications, such as the Blender modelling and animation editor (*http://www.blender3d.org*) and also a number of games and game engines [Daw02]. While the language syntax may be considered unusual as it only allows grouping of command sequences into blocks through code indentation, it has an easy to use API which simplifies the embedding of Python into applications, although this may not be as easy to do as with other embedded languages, such as Lua [Gar06].

3.2.4 Networking

- RakNet (*http://www.jenkinssoftware.com*)
  Raknet is an C++ Open Source networking game engine supporting voice chat with bindings for PortAudio, DirectSound and FMOD. It has the ability to automatically manage the serialisation, creation and destruction of game objects. The purpose of Raknet is to allow the rapid development of online games. It also supports the conversion of single player games to multiplayer.

- ENet (*http://enet.bespin.org*)
  ENet is a thin, simple and robust network communication layer on top of UDP offering reliable, in-order delivery of packets, although it is not intended to provide high level capabilities, such as lobbying, server discovery, encryption or compression.

4 Teaching with Middleware

As part of a university-wide drive towards attempting to ensure that students are well-versed in practical, proactive problem-solving skills in team settings, the creative computing group in the Department for Computing and the Digital Environment (CDE) of Coventry University's Faculty of Engineering and Computing has adopted Activity-Led Learning (ALL) as a pedagogic model.

ALL is a generalisation of problem-based learning [SBM04] that is student-oriented and focuses on "learning by doing". Students engage in the completion of interrelated sets of activities, modelled on practical, real-world scenarios with a view to fostering their problem-solving and leadership abilities as part of a life-long learning initiative. Students are required to take the lead in engaging with all aspects of their subject, while tutors assume supportive roles in facilitating them. We have already described our formative experiences in applying ALL for computer graphics education [AP09]: here, we elaborate on how middleware can be adopted when teaching with student-oriented learning methodologies, such as ALL and PBL, and why it can be particularly desirable for those methodologies. We start by providing a practical exemplar of how middleware is used in our games course, and continue by describing advantages and potential pitfalls to be avoided.

4.1 Example Student Assignment

After an introduction to various games technologies, illustrated with the aid of middleware API's corresponding to these technologies, as well as game engine architecture [AEMC08], students are tasked with the design and implementation of a rudimentary game engine by combining different middleware systems, such as those detailed in Section 3, within a single prototype. This prototype is used as a basis for creating a simple computer game.

4.2 Assessment by Portfolio

While assessment by portfolio seems desirable, care must be taken in terms of assessment when middleware is being used. Firstly, the work conducted by students is not always clearly and easily separable from functionality provided by third-party components. Separation of the students code base is advised to ease marking, in addition to alleviating other problems. However, sometimes this may not be possible, especially when students alter code relating to the components. In such cases, clear reporting by the student seems crucial. Secondly, integration issues between components, or project build issues, may consume a large proportion of the student's task time, and it is important that these difficulties are reported when work is submitted. In some cases, the integration of two major middleware components may represent a challenging project, and its level of difficulty should not be underestimated by the instructor.

4.3 Advantages and Pitfalls

The use of middleware with a top-down approach can be useful as an implicit goal guide for students, providing an existing structure and exemplars towards creating graphics applications. This is particularly useful for shaping their expectations when they must proactively pursue learning objectives, with the lecturer fulfilling the role of facilitator and when they may not yet have extensive experience in the domain. Given a solid basis in programming, the use of appropriate middleware can provide a suitable starting point for students, rather than the formidable blank screen of an empty project: many of the quality middleware components described here contain demonstrators that show students what a working application is capable of doing. By directly interacting with the capabilities offered and being able to experiment with them, this may provide motivation for students towards overcoming bugs and other difficulties, such as the IDE and build process, to create a finished application.

Middleware may also help expose students to working with large

bodies of code featuring many unknowns. It can familiarise them with the notion that they can use classes and functions without necessarily having to fully understanding their internal structure and detail. Large projects, such as OGRE (see Section 3.1.5), must be approached in a piecewise manner and require the adoption of a black-box mindset that students may not have encountered before, at least in a practical setting. Successful exposure here helps to build students' confidence in working successfully with large systems where there may be many unknowns, and also provides a practical lesson in terms of the need for the adoption of good software engineering practice when programming. However, care must be taken to ensure that exposure to many new issues is gradual, so that students do not become overwhelmed.

Another issue is integration. Often underestimated, the integrative approach of using middleware components helps to vividly demonstrate difficulties behind the integration process, particularly in terms of time-requirement estimation and task difficulty.

Additionally, we have consistently witnessed that a key area of difficulty for students is the project build process when multiple components must be considered. Setting dependencies and other project parameters in IDE's often may cause more problems than programming. In such cases, contingency plans should be created to ensure that problems are alleviated so that students' focus remains firmly set on the learning of new concepts, rather than struggling with peripheral issues.

Furthermore, time and work considerations for teaching staff are worthy of mention. In the context of graphics API's alone, Wilkens found that "using multiple API's places a heavy burden on both instructors and support staff" [Wil01], and it could be argued that this may be exacerbated by using API's for different tasks which need to be combined.

As outlined above, the use of middleware has many potential advantages, but must be approached with care. On one hand, students learn practical issues related to integration, project development, the use of black-box components, the value of good software engineering practices, and most importantly, have the scope to create more appealing and intricate graphical experiments than might otherwise be possible. However, there may be a steep learning curve: we identify IDE and project issues as particular areas of concern to be accounted for. Assessment approaches must also be considered carefully, as should the implied burden for instructors. Nonetheless, if presented in a correct manner to students, we believe that the benefits clearly outweigh the potential disadvantages. The successful completion of a project utilising middleware often provides both a fulfilling sense of achievement for students and a strong incentive for further learning.

5 Conclusions

In this work, we have enumerated a list of middleware accessible for education and relevant to the teaching of computer games and graphics curricula. Although often assumed common knowledge, details regarding available systems that educators can leverage in their classrooms are often not common, particularly with respect to licensing aspects that may limit the potential for their usage. This list does not represent an exhaustive middleware list, but rather those that we have encountered and found useful during the course of our teaching; it could be considered a starting point for those considering the introduction of middleware components into their courses.

Additionally, we described some concerns and considerations when introducing the use of middleware into a curriculum. There are many potential advantages, but care must be taken for a variety of reasons. While students learn practical issues related to integration, project development, the use of black-box components, the value of good software engineering practices, and have the scope to create more appealing and intricate graphical experiments than might otherwise be possible, there may be a steep initial learning curve and extra burden for instructors. Nonetheless the successful completion of a project utilising middleware can provide a fulfilling sense of achievement, a strong incentive for further learning and important practical skills to students required for their careers.

References

ANGEL E., CUNNINGHAM S., SHIRLEY P., SUNG K.: Teaching computer graphics without raster-level algorithms. *SIGCSE Bulletin 38*, 1 (2006), 266–267.

ANDERSON E. F., ENGEL S., MCLOUGHLIN L., COMNINOS P.: The case for research in game engine architecture. In *Future Play '08: Proceedings of the 2008 Conference on Future Play* (2008), pp. 228–231.

ANDERSON E. F.: *On the Definition of Non-Player Character Behaviour for Real-Time Simulated Virtual Environments*. PhD thesis, Bournemouth University, 2008.

ANDERSON E. F., PETERS C. E.: On the provision of a comprehensive computer graphics education in the context of computer games: An activity-led instruction approach. In *Eurographics 2009 - Education Papers* (2009), Domik G., Scateni R., (Eds.), Eurographics Association, pp. 7–14.

BOUVIER D. J.: Assignment: scene graphs in computer graphics courses. In *SIGGRAPH '02: ACM SIGGRAPH 2002 conference abstracts and applications* (2002), pp. 42–45.

CUNNINGHAM S.: Re-inventing the introductory computer graphics course: providing tools for a wider audience. In *GVE '99: Proceedings of the Graphics and Visualization Education Workshop* (1999), pp. 45–50.

CUNNINGHAM S.: Computer graphics in context: an approach to a first course in computer graphics. In *SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 Educators Programme* (2008), pp. 1–4.

DAWSON B.: Game Scripting in Python. In *Proceedings of the 2002 Game Developers Conference* (2002).

DELOURA M.: The engine survey: Technology results. Gamasutra Expert Blogs - available from: http://www.gamasutra.com/blogs/MarkDeLoura/20090316/903/The_Engine_Survey_Technology_Results.php, 2009. [Accessed 08/12/2009].

DISTASIO J., WAY T.: Inclusive computer science education using a ready-made computer game framework. In *ITiCSE '07: Proceedings of the 12th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (2007), pp. 116–120.

FRITSCH D., KADA M.: Visualisation using game engines. In *ISPRS commission 5*. 2004, pp. 621–625.

GARCÉS D.: Scripting Language Survey. In *Game Programming Gems 6*. Charles River Media, 2006, pp. 323–340.

HARMON M.: Building Lua into Games. In *Game Programming Gems 5*. Charles River Media, 2005, pp. 115–128.

HITCHNER L. E., SOWIZRAL H. A.: Adapting computer graphics curricula to changes in graphics. *Computers & Graphics 24*, 2 (2000), 283–288.

IERUSALEMSCHY R., DE FIGUEIREDO L. H., CELES W.: Lua - an Extensible Extension Language. *Software: Practice & Experience 26*, 6 (1996), 635–652.

JONES R. M.: Design and implementation of computer games: A capstone course for undergraduate computer science education. In *SIGCSE '00: Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education* (2000), pp. 260–264.

KADA M., ROETTGER S., WEISS K., ERTL T., FRITSCH D.: Real-time visualization of urban landscapes using open-source software. In *Proceedings of ACRS '03* (2003).

LOKI SOFTWARE, INC, HALL J. R.: *Programming Linux Games*. No Starch Press, 2001.

LEWIS M., JACOBSON J.: Game engines in scientific research. *Communications of the ACM 45*, 1 (2002), 27–31.

PARBERRY I., NUNN J. R., SCHEINBERG J., CARSON E., COLE J.: Sage: A simple academic game engine. In *Proceedings of GDCSE'07* (2007), pp. 90–94.

SAVIN-BADEN M., MAJOR C.: *Foundations of Problem Based Learning*. Open University Press, 2004.

SUNG K., SHIRLEY P., ROSENBERG B. R.: Experiencing aspects of games programming in an introductory computer graphics class. *SIGCSE Bulletin 39*, 1 (2007), 249–253.

WILKENS L.: A multi-api course in computer graphics. *Journal of Computing Sciences in Colleges 16*, 4 (2001), 66–73.

WAGNER D., KAINZ B., SCHMALSTIEG D.: Realtime 3d graphics programming using the quake3 engine. CGEMS: Computer Graphics Educational Materials Source, The CGEMS Project, 2008.