Randomly Generated 3D Environments for Serious Games

Jeremy Noghani
Interactive Worlds Applied Research Group
Coventry University
Coventry, UK
noghanij@coventry.ac.uk

Fotis Liarokapis
Interactive Worlds Applied Research Group
Coventry University
Coventry, UK
F.Liarokapis@coventry.ac.uk

Eike Falk Anderson
Interactive Worlds Applied Research Group
Coventry University
Coventry, UK
Eike.Anderson@coventry.ac.uk

Abstract— This paper describes a variety of methods that can be used to create realistic, random 3D environments for serious games requiring real-time performance. These include the generation of terrain, vegetation and building structures. An interactive flight simulator has been created as proof of concept. An initial evaluation with two small samples of users (remote and hallway) revealed some usability issues but also showed that overall the flight simulator is enjoyable and appears realistic and believable.

Keywords – serious games; 3D terrain modeling; computer graphics; flight simulator.

I. INTRODUCTION

The creation of realistic virtual environments is an important issue in the computer animation, computer games, digital film effects and simulation industries. In recent years, the computer and video games industry has overtaken both the film and music industries as the top revenue producers, and the cost for developing a commercial game now usually requires investments of several million dollars, involving large teams of developers that can number in the hundreds of workers, many of whom are artists and designers providing content for the decoration of rich virtual game worlds. While many games companies have the necessary budget to develop these expensive modern computer games that employ state of the art computer graphics, not all game developers have the same resources. Serious games refer to computer games that are not limited to the aim of providing just entertainment but which can be used for other purposes, such as education or training in a number of application domains. There are several game engines and online virtual environments that have been used to design and implement these games for non-leisure purposes [1]. The development of serious games using the same approach as used for entertainment games is not possible because their budget is usually limited to a few thousand dollars. The literature states that when games and simulations technologies are applied to non-entertainment domains, serious gaming applications can be created [2]. When classifying a game, the definition of the term ‘game’ does not necessarily require formalised criteria for success such as praising winners, totalling points or reaching certain areas in a level [3]. “Gaming is by no means a replacement for existing model and simulation building processes and practices but it has tangible advantages that ultimately could result in wider, more flexible, and more versatile products” [4].

To overcome these problems, a variety of methods for automatically creating detailed but also randomised environments have been developed. The use of these procedural methods [5] saves time and reduces the budget for creating effective serious games. However, if a user wishes to interact with the environment in a meaningful way, such as in a flight simulator that has an expansive world and implements collision detection, then numerous problems arise that are often not dealt with during the creation stage.

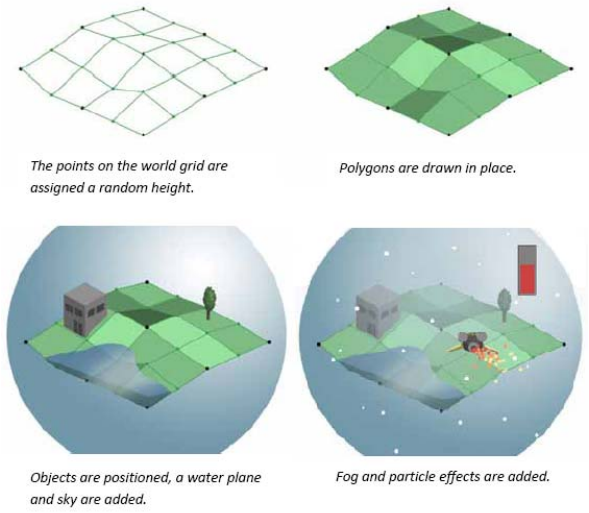

Figure 1 Randomly generated environment for an entertainment flight simulator

This paper explains some of the problems that can arise from this situation and describes a variety of methods that can be used to overcome them. These methods have been applied to a basic flight simulator (see Figure 1), so that the results could be observed and evaluated. Initial results with 2 types of user groups (remote and hallway) revealed some usuablity issues but also illustrated that overall the flight simulator is enjoyable, fun and looks realistic.

The rest of the paper is structured as follows. Section II provides past methods used in terrain generation. Section III presents how our flight simulator serious game was created to allow for navigation and interaction with the terrain whereas section IV describes

techniques used for creating infinite terrains. Section V provides an overview of procedural techniques for adding vegetation and buildings into randomised environments. Section VII presents a flight simulator as a case study and section VIII illustrates initial evaluation results. Finally section IX presents conclusions and future work.

II. TERRAIN CREATION METHODS

The majority of the traditional methods used to create partially randomised terrain involve the use of fractals, such as fault formation [6] and noise algorithms [7]. Fractals are objects or shapes which, when split into smaller parts, result in shapes that are similar to the original shape as a whole [8] (self-similarity). Their use is advantageous from a computer graphics point of view, due to their ability to define complex geometry from a small set of instructions, and due to their ability to define shapes that are often difficult to define with simple Euclidian geometry.

Random one-dimensional midpoint displacement is a simple algorithm that can be used to create fractals that appear similar to the two-dimensional silhouette of mountain ranges. It is implemented by finding the midpoint of a single line, and displacing its height by a random offset value. This process is then repeated at the midpoints between these newly defined points with a reduced random number range. This algorithm is usually implemented recursively to allow the silhouette to be made as detailed as the user requires [9].

When the random midpoint displacement is applied to the centre of a terrain grid square only, this can be defined in terms of the displacements of the centre points of the square's sides. A more efficient way is to derive the same result by adding the four corners of the square and dividing them by four and adding the random value to the result.

The diamond-square algorithm can be considered an effective way of applying this one-dimensional method to a second dimension, creating three-dimensional terrain if the resulting lattice is used as a virtual heightmap [10]. The recursive algorithm works by refining a square area, whose four corner points' height values may be initialised randomly, and then calculating its centre point by calculating a mean of the corner points, to which a random value is added. Midpoints of the edges between the corners are then calculated in a similar manner and the original shape is then subdivided by generating new edges between the newly generated points, forming new squares for further subdivision, as well as diamond shapes within the squares. Using a smaller random value tends to result in the creation of smoother terrain, whereas larger offsets result in more jaggered edges. The use of hexagonal and triangular shapes instead of a square grid has been proposed to reduce the problems of 'creasing' in the terrain [11].

There has been some work on modelling terrain based on realistic physical constraints. Kelley et al. [12] produced a system in which water drainage is simulated to shape and constrain the landscape, in a similar manner to the way in which water erosion affects real terrain. Musgrave et al. [13] managed to achieve realistic results through a different method that took hydraulic and thermal erosion into account when creating a fractal terrain. Attempts to create more geographically accurate models have lead to increased realism in some aspects, but have also increased the complexity of the design and rendering [14].

An alternative method to create randomised terrain is the use of Lindenmayer systems. L-systems were originally created to study organic growth, such as is the case with plants, but they can easily be adapted to cover other self-similar structures, such as mountainous terrain [15]. The distinctive feature of L-systems is the use of rules that rewrite strings, which can be called recursively to make a hierarchy of strings. When displayed visually, these may produce results similar to those of a mountain range silhouette, for example.

III. SIMULATOR CREATION

For the purpose of this paper, a small, simple flight simulator was created to allow for navigation and interaction with the terrain. In the design, usability took precedence over realism, as a result of which the controls were deliberately kept simple; the mouse is used to alter the yaw and pitch of the aircraft, and two keys are used for acceleration and deceleration. Additionally, to allow for close examination of the terrain, the in-program physics were kept liberal; i.e. the plane was allowed to come to a complete stop in mid-air without gravity taking effect.

For the terrain itself, heightmaps that were generated using the diamond-square algorithm were chosen to provide surface detail. This method was chosen primarily due to the algorithm's simplicity and adaptability, meaning that the system itself could easily be altered to accommodate a more complex algorithm or to accept alterations to factors such as surface roughness without the need to be rewritten from scratch. Additionally, by choosing a recursive algorithm, the level of detail could be adjusted as necessary, which proved to be an advantage when dealing with different methods that required different levels of processing power.

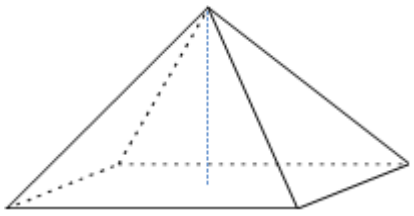

Figure 2 Pyramid created by a random height-displacement of the centre point of the square base

Figure 2 illustrates how a vertical deformation of the centre of the square base connected to the neighbouring points can produce a pyramid. Instead of using this linear deformation of the neighbouring points, a two-dimensional Lorentz distribution [16] for the

height array was assumed. By adjusting the width of the Lorentzian shape, one could obtain a means for controlling the smoothness of the terrain. Another distribution that could be used is the Gaussian shape. After some trials, it was found that this bell-shaped distribution that creates a smooth terrain was the following:

$$height = y = (random\ number) \times \frac{D^2/8}{(x-x_o)^2 + (z-z_o)^2 + D^2/8}$$

where (x_0, z_0) is the position of the peak and D^2 is the length of the square which is related with the width of the Lorentz distribution. The value of the width directly affects the smoothness of the terrain. By decreasing the width of the bell-shaped distribution, the terrain becomes steeper.

Water was added to the landscape in the form of a single translucent plane placed at an appropriate height (see Figure 1). Small buildings and trees were also added as decorations for the terrain, using pre-fabricated models. Their placement on the landscape was decided randomly, although rules were implemented to prevent their creation on top of mountain peaks, below the virtual world's water level, or on steep slopes.

IV. OUTER BOUNDARIES

In any simulator that requires travelling for a long time in one direction (flight simulators being the most notable example), the user may find that they see or pass an "outer boundary"; the terrain is only created within certain dimensions, so reaching an area where nothing is rendered can be a possibility, depending on the actual implementation. Three potential solutions were devised and implemented.

The first was to create a terrain of such large extent that the user would never reach the outer boundary (see Figure 3). The success of this method depended upon the scale of the landscape relative to the user's movement speed; if a user moved over the length of single terrain grid squares per second, then they would be less likely to reach a boundary than a user who moved at 5 grid squares per second, assuming that other factors remained equal. The implication of this is that landscapes must be scaled to be as large as possible to minimise the chance of the user discovering a boundary. Upon attempting this method, another problem arose in the form of the terrain looking bland and flat, due to the spread-out nature of the polygons. The solution to this was to increase the number of subdivisions, thus increasing the level of detail. However, it ought to be noted that if the designer were to keep increasing the scale and the level of detail of the terrain at the same time, then eventually the computer would reach the limitations of its hardware. For this reason, this method can be considered appropriate for small demonstration purposes or applications where the user moves slowly relative to the virtual landscape, but inappropriate for a full flight simulator where a large detailed environment is desirable.

The second attempted method was to "loop" the old landscape when the user reaches a boundary. In order for the landscape to loop seamlessly, it was vital to ensure that the edges have equal height values; on an A1 to H9 grid, for example, B1 must equal B9, A5 must equal G5, and the corner values (A1, A9, G1, and H9) must equal each other. The landscape can then be kept in the computer memory as a single "tile", which can be duplicated to a new spot when needed. Tiles of terrain that are a particular distance from the player's avatar, i.e. the aircraft, can be deleted or moved to a more appropriate position, to keep memory usage to a minimum.

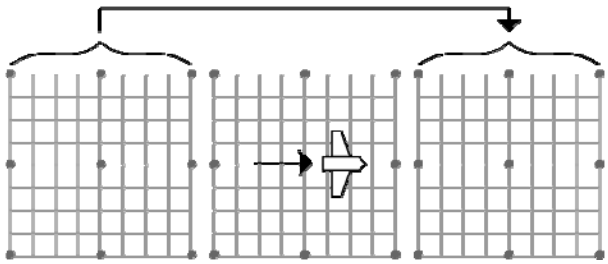

Figure 3 Upon reaching the right side of the landscape, a tile of terrain is moved in front of the player to provide the illusion of endless terrain

This solution could cause the problem of the user noticing the repetitive nature of the terrain (especially if the terrain contained a notable feature, such as a peak), but the severity of this issue would depend on several factors. For example, if the user intended on using the same terrain for a long period of time, then he or she would be more likely to notice the copied terrain tiles than a user who intended on playing for a short period of time. Additionally, if the terrain is seemingly large (i.e. if it took 60 seconds to travel across a single tile, for instance), then the repeating nature of the terrain tiles would be less noticeable than if the terrain were small (i.e. if it took 20 seconds to travel across a tile). The implications of this are that looping the landscape with the same terrain tile would work in a number of simulation scenarios, but it is difficult to assess whether this could be successfully applied to a particular flying simulator without some form of user testing. A final note on this method is that by making the four corner values identical, the deviation between the highest and lowest points of the landscape may be reduced. Alternate methods of increasing the stochasticity (such as using more random points) may be considered to avoid this problem.

The final method was to automatically generate a new, unique tile of terrain when the user reaches a boundary (see Figure 4). To ensure that the tiles matched seamlessly, one edge of the terrain would have its heights copied to the matching edge of the new tile. The rest of the heights can then be calculated via random values and midpoint displacement using the diamond-

square algorithm, in the same manner as was used to create the first tile.

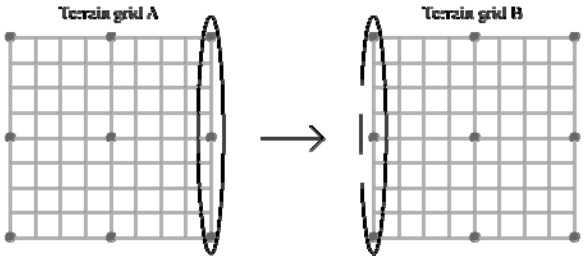

Figure 4 Upon reaching the right side of terrain grid A, the values of the far-right points are copied to the far left points of terrain grid B. The other points of terrain grid B are then calculated via randomisation and mid-point displacement, as done for grid A

The advantage of this method is that the terrain is genuinely infinite; from a user's perspective, the land would continue in all directions with no repetitions. However, there is the problem of memory usage. If a user were to continue travelling in a straight line, increasingly more tiles would have to be generated and stored in memory (even if they were not rendered), which could cause a memory overflow. The solutions to this are to either store previously visited tiles in a separate cache file, or to simply delete previously visited areas that are far away from the avatar. The former solution has the problem of requiring an efficient caching system (i.e. one that allows for fast writing and reading of large sets of coordinates), and the latter suffers from not allowing the user to backtrack to a previously-visited area that is a certain distance away.

V. Automatic Decoration of Virtual Environments

Empty, featureless spaces resulting from terrain generation alone are insufficient for the creation of convincing virtual environments. To overcome this, the environment needs to be decorated with suitable vegetation and artificial structures, including buildings as well as settlements.

A. Procedural Generation and Placement of Vegetation

There are different methods for the procedural creation of vegetation, many of which are based on fractal or simpler rule-based techniques. One of the latter methods has been used for on-the-fly generation of forests for real-time virtual environments [17], using a skeletal topology for procedurally generated and animated trees, which has also been combined effectively with on-the-fly generated grass to create a rich natural scenery [18]. A much more powerful, rule-based approach applying component-based modelling is the one by Lintermann and Deussen [19], which provides a more intuitive way for controlling plant modelling than the well-known L-Systems [15]. The decision of what type of plant needs to be placed into the virtual world usually depends on a number of factors, including the elevation and slope of the terrain, as well as topographic features that dictate the probability of a specific plant's occurrence [20]. Once a position in the generated terrain has been decided, the generation of plant models can be followed with the placement of the vegetation.

For the proof of concept application, a simplified method was implemented that made use of randomised positioning of vegetation models, rather than random vegetation itself. A series of low-polygon trees and bushes were created and exported as 3D models, which were then loaded at the start of the program. Once the terrain had been created, the vegetation was randomly assigned to various places on the terrain grid. However, if a particular part of the terrain was too high, low, submerged in water, or on a heavily inclined slope, then that area was rejected and ignored, the purpose being to prevent vegetation appearing in unrealistic locations. To reduce the problem of repetition, the vegetation was also rotated and scaled by a random value. The result was surprisingly effective; the plants appeared to have stochastic properties despite being pre-defined models, and it was only when the density of the vegetation was increased to the level of woodland when the repetition became noticeable.

B. Procedural Generation of Buildings

Most real-world environments include some sort of artificial structures. In a rural setting these might be scattered houses that make up only a fraction of the decorations of the terrain, with the majority of decorations being plants, whereas in urban settings this would be reversed with buildings providing the majority of virtual world decorations. If the level of detail required for buildings is relatively low, as would be the case in a flight simulator that depicts the virtual world from a high altitude, then simple geometric bodies can create adequate results if combined with suitable texture maps that hide the lack of actual detail in the geometry. The use of 'split grammars' [21] and 'shape grammars' [22] for describing architectural features allow the use of much more complex shapes and building structures, which can be intricately detailed [23].

At the greatest level of detail, even building interiors can be generated [24]. The placement of these artificial structures in the virtual world can reach great levels of complexity if the buildings form part of an urban environment [25]. These more complex settlements are created in a series of steps [26]: (a) *first a suitable road network is generated, effectively providing street maps that partition the terrain and to constrain the placement of buildings* (b*) this is then used to direct the division of the terrain into lots which may be partitioned further to generate building footprints*, and (c) *which are then used as input for the generation of the buildings themselves*.

Due to time restrictions, attempts at implementing a more complex urban generation system had to be simplified. The method of distributing buildings was therefore nearly identical to the method of distributing

plants, with a few distinct changes. Firstly, the building models were adjusted to have 'foundations', or basement levels. The purpose of this was to prevent the underside of the model showing, should the building be positioned on a slope. Secondly, rather than being rotated and scaled, which would be inappropriate for the majority of buildings, structures were assigned random 'height' and 'extention' values that copied parts of the building model above or to the side of the original, the purpose being to reduce repetition and to reduce the isolated feel that can be associated with solitary buildings.

The results were acceptable, and would be especially fitting if applied to a small settlement of village size, but the environment as a whole lacked the structure and density associated with urban areas. The solution to this problem would be to redesign the placement system, and possibly the terrain generation system, from scratch, whilst taking into account the architectural shape grammars and road networks used in previous city generation applications.

VI. COLLISION DETECTION

Accurate detection of when an object hits the terrain surface is highly desirable in many applications, especially flight simulators. However, calculating precise polygon overlap between an aircraft and a landscape would be too computationally expensive, especially given the recursive and detailed nature of fractal terrain, which could result in thousands of checks per frame. Alternative methods were therefore implemented in an attempt to find a method that was both fast and accurate.

Traditionally, the most common method of calculating collisions is through the use of bounding volumes, such as spheres or orientated cuboids, which can be positioned in place of a complex model and then be checked for overlap. They can also be used hierarchally (i.e. checking a large bounding volume, followed by more precise checks), to give results that are both memory efficient and precise [27]. In this paper, a single axis-aligned bounding box was used to cover the aircraft. The trees, vegetation and buildings were covered by single oriented bounding boxes. Additionally, a single plane check was used to check whether the aircraft had hit the water surface. More complex and accurate methods, such as a series of bounding boxes, would be more appropriate for a final game or simulation, but simplicity was adhered to for the sake of shortening the debugging process and for achieving consistent results when testing the speed and accuracy of the collisions.

For the terrain in this instance, bounding spheres were quickly deemed as inappropriate, as they would fit awkwardly with the relatively flat polygons unless used at a high level of detail. Axis-aligned bounding boxes would be faster than spheres to calculate, due to the simpler calculations needed for every frame [27], and they could potentially be more accurate on less mountainous terrain due to the nearly-aligned nature of the landscape. The bounding boxes were applied by making use of the terrain data arrays; for every polygon point, a box was applied that would match the point's height, and have a polygon's length and width. An advantage of this method was that, as with the fractal terrain itself, the checks could be called recursively to achieve a higher level of detail (collision accuracy), at the cost of efficiency.

For example, a bounding box could be applied every two polygons across, or bounding boxes of twice the width and length could be applied every four polygons across, resulting in significantly fewer calculations. During testing, it was found that, relative to the complexity of the fractal terrain, only a small number of bounding boxes were needed for the terrain collisions to be perceived as accurate, so program efficiency was not an issue. More bounding boxes were needed to maintain accuracy if the terrain was made to be mountainous. However, a scenario where more bounding boxes were needed than were possible with the terrain data array was not implemented. One observed point was that mountain peaks seemed to suffer occasionally from inaccurate collisions, probably due to the fact that this was the area with the most 'space' contained within the box. There did not seem to be a simple fix to this problem, as manually adjusting the bounding boxes used for mountain peaks was impractical and delivered mixed results. Nonetheless, axis-aligned bounding boxes were considered successful for this simulator due to their speed and overall accuracy.

Before oriented bounding boxes can be discussed, it ought to be noted that since we are working in three dimensions, there are two sets of rotation that can be implemented. The first would align the boxes according to the way the landscape is facing (the yaw); this would appear to be a rotated square, if viewed from above. The second would align the boxes according to the slope of a polygon or terrain face (the pitch and roll). Performing a check on whether an axis-aligned box (the aircraft) collides with a rotated square (part of the terrain after being rotated once) would require a simple series of checks for each of the oriented box's points; the collision detection is still being performed in two dimensions. However, after applying the second rotation, checks must be performed between two bounding boxes aligned on separate axes, and consequently the number of calculations rises steeply. Considering the number of bounding boxes on the terrain, it was predicted that this could potentially become an issue.

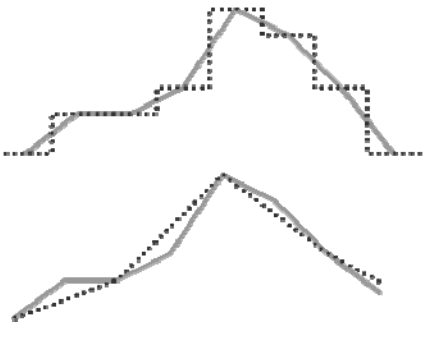

Figure 5 Comparison of axis-aligned (top) and oriented bounding box method (bottom). Recursive calls lead to a more accurate terrain match.

Upon testing, it was found that performing only a single rotation on the bounding boxes gave very similar results to the axis aligned boxes, both in terms of efficiency and accuracy. This method carried the same problems and advantages of the axis aligned method. Upon rotating the boxes a second time, the accuracy improved somewhat and the problems associated with the mountain peak inaccuracies disappeared. However, the program was notably less efficient; attempting to apply the fully oriented bounding boxes to every terrain polygon slowed the program down to an unusable level (although whether such a level of accuracy is actually required for a flight simulator is questionable). Deciding whether it would be more accurate and efficient to use a small number of fully oriented bounding boxes or a large number of axis-aligned boxes for fractal terrain is a matter that requires further research and testing.

The program could be further streamlined through the addition of Bounding Volume Hierarchies (BVHs). By splitting up the terrain area into large bounding volumes that in turn contain consecutively smaller bounding volumes, the number of collision checks carried out per frame could be substantially reduced. Since this particular program uses a square grid for the terrain, it would be logical for the checks to take the form of testing which side the aircraft falls on on an imaginary plane placed in the middle of the terrain grid. This is repeated to further divide the grid into quarters and eighths within the section in which the craft is located. Precise collision checks can then be carried out in the appropriate section. In this case, the efficiency of using BVHs depends upon the complexity of the landscape; a more complex landscape would benefit more from having BVHs implemented for collision detection, as a large number of calculations would be removed per frame, whereas if the landscape were only a few polygons in size, implementing BVHs would have little effect. If numerous aircraft are included, or buildings and objects overlap BVH boundaries, then the efficiency of BVHs becomes more difficult to calculate, and their use ought to be carefully considered.

VII. FLIGHT SIMULATOR

The simple flight simulator game created as a case study for the random world generation allows navigation and interaction with the randomised 3D environment. Thematically the game is set to take place on alien worlds. This could potentially offer a wider range of potential scenarios (i.e. the online virtual world Second Life is based on this philosophy) and thus offer a higher level of entertainment and enjoyability compared to 'Earth' based scenarios. While of course this type of representation does not automatically lead to exploratory, challenging and problem-based learning experiences, the opportunities for players to "define their learning experiences or pathways, using the virtual mediations within virtual worlds, has the potential to invert the more hierarchical relationships associated with traditional learning, thereby leading to more learner-led approaches based upon activities for example" [28].

Players can navigate intuitively inside the the alien worlds using mouse and keyboard inputs. As mentioned before, the controls were deliberately kept simple; the mouse is used to steer the aircraft, and two keys, 'W' and 'S', are used for acceleration and deceleration respectively. Players can also fire a weapon by pointing and clicking with the mouse. An overview of the flight simulator in operation is shown in Figure 6.

Figure 6 Alien world flight simulator in operation

A digital compass and a speedometer are implemented os overlays in the interface. The digital compass (see top left hand side of Figure 6) allows players to navigate inside the virtual environment using directional information. The speedometer representation (see top right hand side of Figure 6) was kept as simple as possible in order to leave maximum screen space available for the game. Additionally, a widget menu was implemented that allows players to change specific components of the 3D environment by right-clicking the mouse. The environmental components that can be modified include: *terrain re-generation*, *weather alteration (i.e. rain, fog, sunny, etc.)*, and *colouring of the grass, water and sky*. An overview of the widget menu is shown in Figure 7.

Figure 7 Flight simulator widget menu

Options to change the music track or mute it entirely were also added to the widget menu. These controls were introduced to the user in the form of a simple menu

screen that appeared at the start of the game. The collision detection algorithm was based on axis-aligned bounding boxes for the terrain and the building structures (see section VI). In addition, based on the techniques described in section IV, the landscape creates the illusion that is infinite. An example screenshot of explosions generated when the aircraft collides with the ground is shown in Figure 8.

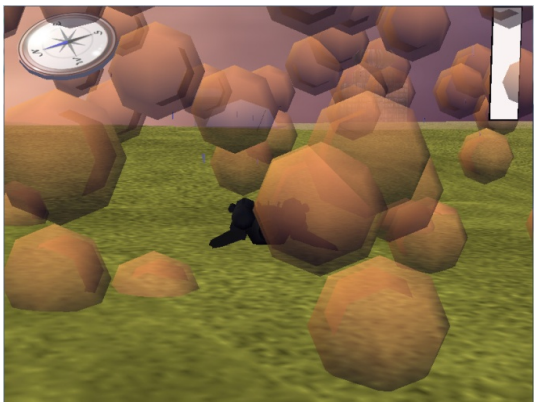

Figure 8 Collision between the aircraft and the terrain

To simulate the effect of collisions, explosions were incorporated into the game based on particle systems. Each particle effect (snow, rain, engine fumes and explosions) has a velocity value, rotation value, or transparency applied to it and for each collision, each particle is changed appropriately (such as a slight decrease in y position, for rain), and if it reaches a certain condition (such as rain falling too low), it is set to a new start position.

VIII. Initial Evaluation

To acquire feedback on the finished core of the application, a self-contained executable file was supplied to two sets of users: a small Internet general discussion forum (remote usability testing), and a group of Coventry University students (hallway usability testing). The intention of these tests was primarily to gather information on the playability and enjoyability of the game, but also to discover potential technical problems. All of the end-users had some experience with games, and the vast majority described themselves as 'gamers'. A few of those involved also had experience with games programming, or had some knowledge of the architecture behind creating a game. For both sets of users, the aim of the flight simulator project was presented and it was explained that the players should not expect a complete game, but rather a prototype.

A. Remote Testing

For the Internet forum users, the following set of questions was provided: (a) *do the collisions seem accurate?* (b) *would you be interested in playing a fuller version of this game?*, (c) *what would you like to see added? (e.g. larger variety of landscapes, different controls)* and (d) *how large and varied were the environments?* – the final question testing, whether the attempts at making the different environments varied was a success. A qualitative analysis was done with five users. Feedback was received some in direct reply to the questions, and some raising additional issues. Recorded feedback was very encouraging and all users agreed that the methods used were very useful for the creation of serious games applications, although important issues were pointed out.

The answers to the second and third questions were positive and similar. Several users commented that they would play such a game, on the condition that further additions were made to the gameplay. "It needs more to do but the engine is cool", one user noted. In regards to the final question, reactions were mixed. One user commented that they "enjoyed exploring the worlds", and mentioned how the use of colour made a lot of difference, but another noted that "the buildings look samey. They need more variation". Additional comments were also made. One player complimented the "atmosphere" of the game. However, the collision detection was criticised by another player. Specifically, he stated "I sometimes crash when I drive too close to mountains. [The collision detection] is fine for the water and flat land though". The other users claimed that the collision detection was acceptable.

B. Hallway Testing

Four students from the Faculty of Engineering and Computing, Coventry University were asked to partake in the second test group. Instead of asking the university students questions, they were asked to talk through what they were doing and how they felt as they played the game. Two students had some issues with the controls. Specifically, they found the delay between hitting a key and the aircraft movement difficult to get to grips with. It is worth-mentioning that after playing for long enough, the players adjusted to the issue. The object 'popping' due to terrain decorations being added to the scene was criticised by two players; one commented that "it's nice that I can keep going forever, but it's annoying that I can't see the horizon properly". Similar to the Internet test group, two players complimented the 'feel' of the virtual worlds. One admired the water and sky effects, and the other spent a fair amount of time recreating different landscapes. Despite making use of the repeated tile method for this test, none of the users were aware of the repetition, which meant that this method is successful and efficient for terrains that are explored by the user for less than five minutes. Further testing would be needed for the effectiveness of this method over longer periods of time, however.

IX. Conclusions and Future Work

This paper discussed a number of methods that can be used to create realistic but also randomised 3D environments for serious games. These methods referred

to the automated generation of fractal terrains, vegetation and building structures. To prove the feasibility of the techniques, an interactive flight simulator has been implemented and evaluated. Initial results with two different types of user groups (remote and hallway) showed that overall the flight simulator is enjoyable, looks realistic for a gaming scenario and thus also has the potential to be used for the development of serious games.

In the future, a classification regarding buildings and vegetation will be developed allowing for automatic random generation of larger urban environments. To improve the cognitive perception of the players, additional urban geometry will be generated automatically in the game such as: streets, pavements, signs etc., similar to the framework proposed by Smelik et al. [28]. Finally, scenarios will be developed and more evaluation studies will be performed with more users.

ACKNOWLEDGEMENTS

The authors would like to thank the Interactive Worlds Applied Research Group (iWARG) members for their their support and inspiration. A video that illustrates the application in action can be found at: http://www.youtube.com/watch?v=6G1NSALgSEY

REFERENCES

[1] Anderson, E.F., McLoughlin, L., Liarokapis, F., Peters, C., Petridis, P., de Freitas, S. Serious Games in Cultural Heritage, Proc. of the 10th Int'l Symposium on Virtual Reality, Archaeology and Cultural Heritage, VAST-STAR, Short and Project Proceedings, Eurographics, Malta, 22-25 September, 29-48, (2009).

[2] Zyda, M. From visual simulation to virtual reality to games. IEEE Computer 38(9): 25-32, (2005).

[3] Krause, D. Serious Games – The State of the Game, The relationship between virtual worlds and Web 3D, White Paper, Pixelpark Agentur, (2008).

[4] Sawyer, B. Serious Games: Improving Public Policy through Game-based Learning and Simulation, Foresight and Governance Project, Available at: [http://www.seriousgames.org/images/seriousarticle.pdf], Accessed at: 26/10/2009.

[5] Smelik, R.M., de Kraker, K.J., Groenewegen, S.A., Tutenel, T., Bidarra, R. A Survey of Procedural Methods for Terrain Modelling, Proc. of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS), (2009).

[6] Shankel. J. Fractal Terrain generation – Fault Formation, Game Programming Gems, Charles River Media, 499-502, (2000).

[7] Perlin, K. An Image Synthesizer. Proc. of ACM SIGGRAPH '85, 287-296, (1985).

[8] Mandelbrot, B. The Fractal Geometry of Nature, W H Freeman, New York, (1983).

[9] Fournier, A., Fussel, D., Carpenter, L. Computer Rendering of Stochastic Models. Communications of the ACM, 25(6): 371-384, (1982).

[10] Miller, G.S.P. The definition and rendering of terrain maps, Proc. of ACM SIGGRAPH '86, ACM Press, 39-48, (1986).

[11] Peitgen, H. Saupe, D. The Science of Fractal Images, Springer-Verlag, (1998).

[12] Kelley, A.D., Malin, M.C., Nielson, G.M. Terrain simulation using a model of stream erosion. Proc. of ACM SIGGRAPH '88, ACM Press, 263-268, (1988).

[13] Musgrave, F.K., Kolb, C.E., Mace, R.S. The synthesis and rendering of eroded fractal terrains. Computer Graphics 23(3): 41-50, (1989).

[14] Belhadj, F. Terrain Modeling: A Constrained Fractal Model. Proc. of the 5th Int'l Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa, Grahamstown, South Africa, 197-204, (2007).

[15] Prusinkiewicz, P., Lindenmayer, A. The Algorithmic Beauty of Plants, Springer-Verlag New York, Inc, (1990).

[16] Hecht, E. Optics, 2nd Edition, Addison Wesley, 603, (1987).

[17] Di Giacomo, T., Capo, S. and Faure, F. An interactive forest. Proc. of the 2001 Eurographics Workshop on Computer Animation and Simulation, 65-74, (2001).

[18] Guerraz, S., Perbet, F., Raulo, D., Faure, F., and Cani, M-P. A Procedural Approach to Animate Interactive Natural Sceneries. Proc. of the 16th Int'l Conference on Computer Animation and Social Agents (CASA 2003), 73-78, (2003).

[19] Lintermann, B. and Deussen, O. Interactive Modeling of Plants. IEEE Computer Graphics and Applications 19 (1): 56-65, (1999).

[20] Wells, W.D. Generating Enhanced Natural Environments and Terrain for Interactive Combat Simulations. Doctoral Dissertation, Naval Postgraduate School, Monterey (CA), (2005).

[21] Wonka, P., Wimmer, M., Sillion, F., Ribarsky, W. Instant Architecture. ACM Transactions on Graphics 22(3): 669-677, (2003).

[22] Müller, P. Wonka, P., Haegler, S., Ulmer, A. Van Gool, L. Procedural Modeling of Buildings. Proc. of ACM SIGGRAPH 2006, ACM Press, 614-623, (2006).

[23] Havemann, S. Generative Mesh Modelling. Doctoral Dissertation, Technische Universität Braunschweig, (2005).

[24] Hahn, E., Bose, P., Whitehead, A. Persistent Realtime Building Interior Generation. Proc. of Sandbox Symposium 2006, 179-186, (2006).

[25] Greuter, S., Parker, J., Stewart, N., Leach, G. Real-time Procedural Generation of 'Pseudo Infinite' Cities. Proc. of GRAPHITE 2003, ACM SIGGRAPH, 87-94, (2003).

[26] Parish, Y.I.H., Müller, P. Procedural Modeling of Cities. Proc. of ACM SIGGRAPH 2001, ACM Press, 301-308, (2001).

[27] Gottschalk, S., Lin, M.C., Manocha, D. OBBTree: A Hierarchical Structure for Rapid Interference Detection, Proc. of SIGGRAPH '96, ACM Press, 171-180, (1996).

[28] De Freitas, S. Serious Virtual Worlds - A scoping study, JISC, (2008), Available at: [http://www.jisc.ac.uk/media/documents/publications/seriousvirtualworldsv1.pdf], Accessed at: 26/10/2009.

[29] Smelik, R.M., Tutenel. T., de Kraker, K.J., Bidarra, R. A procedural Terrain Modelling Framework, Poster Proc. of the Eurographics Symposium on Virtual Environments EGVE08, 39-42, (2008).