Honours Project Report

Automatic Creation of Physics Components for Procedurally Generated Trees

Zacharia Crumley

Supervised By:

Dr. James Gain

Rudy Neeser

	Category	**Min**	**Max**	**Chosen**
1	Software Engineering/System Analysis	0	15	0
2	Theoretical Analysis	0	25	0
3	Experiment Design and Execution	0	20	10
4	System Development and Implementation	0	15	15
5	Results, Findings and Conclusion	10	20	15
6	Aim Formulation and Background Work	10		10
7	Quality of Report Writing and Presentation	10		10
8	Adherence to Project Proposal and Quality of Deliverables	10		10
9	Overall General Project Evaluation	0	10	10
Total marks		**80**		**80**

Department of Computer Science

University of Cape Town

2009

Abstract

The field of procedural generation is becoming increasingly widespread as a tool for generating content in video games and virtual environments. Procedural generation creates content (such as 3D models) automatically, saving large amounts of time and money, as no human interaction is required. One area where procedural generation excels is the creation of trees, which is most commonly achieved using L-systems.

The problem with these generated trees is that they are static and do not have any associated animations or dynamic behaviour. Our research addresses this problem by presenting a technique for adding physics components to trees created by L-systems, so that they can be simulated using a real-time physics engine, making them react dynamically, and realistically, to their environment. The algorithm works by creating a physics skeleton for the tree that can be used to approximate real tree behaviour.

In order to improve the computational performance of our system, we also present some level-of-detail schemes aimed at real-time physics simulations. These schemes are not suitable for all cases, especially those which require a high level of realism, but can improve performance.

We test both the computational performance of our system, including it's ability to operate in real-time, as well as the realism shown in the dynamics of the produced trees. The results indicate that our trees are very realistic and an improvement over those found in modern video games. However they also show that our system is not suitable for real-time use as it stands. We believe that research into further optimizations and culling techniques could change this, and recommend a number of avenues for future research in this regard.

Table of Contents

Illustration Index

1 Introduction

This research project is aimed at investigating ways to enhance procedurally generated trees so that they behave dynamically and interact realistically with the environment around them. Procedurally generated trees are often used in the scenery found in video games and virtual environments[1] and can be very important in enhancing the realism and immersion that users experience when interacting with those systems. Traditionally these trees have been immobile and static, which is problematic since real trees do not behave in this manner, meaning that users are likely to notice the immobile nature of the trees. This can break their immersion and decrease the perceived realism of the game or virtual environment.

For this reason, we conducted research into ways to solve this problem and create dynamic procedurally created trees that realistically interact with the environment around them.

1.1 Procedural Generation

In the field of computer graphics and virtual environments, generating large amounts of detailed content (3D models, textures, environments, etc.) has been a problem, and an area of active research, for some time [33,39]. Traditionally, this content has been created "by hand" by professional modellers and artists using specialized software. The upside to this approach is that created content is generally (but not always) detailed, of high quality, and appropriate for whatever use it was intended. The downside is that the process is slow and expensive, since creating the content takes time (proportional to the amount of content, and it's quality) and the employment of teams of artists and modellers is costly. In the majority of cases, this is how content is currently created, although, for some types of content, products such as SpeedTree[2] are changing this.

The field of procedural generation [11,32,39,41] aims to solve these problems by automating the creation of models and other visual content. There are many areas in procedural generation, covering a wide range of content, with many different methods. The details of the methods vary, but their main similarity is they all require much less human input and interaction, to produce content of an acceptable quality, than more manual methods.

There are a number of techniques in procedural generation, ranging from simple, randomised tables, to more complex methods such as gene expression programming [47] and various types of L-systems [39]. Most of these approaches use seeded pseudo-random sequences in order to produce different results on each run. This is key when you are generating many objects of the same type (for example buildings); it is important for the visual quality of the program that they all look substantially different. Users will almost certainly notice if every building in a scene is exactly the same.

Not all types of models and content can be created via procedural generation (for example, models of animals or humans), and there are definite limits to the technique's usefulness (for

1 A list of some of these games can be found at http://www.speedtree.com/gallery/
2 http://www.speedtree.com/

example, the lack of fine control over the outputs produced by an L-system), but within these limits there are several areas where procedural generation excels. These include the generation of trees and plants [39], music [14], landscapes [31] and clouds [41]. Their performance in these areas coupled with the benefit of requiring little-to-no human interaction (which saves on the time and costs of content creation) means that procedural generation is a valuable and useful technique for 3D content.

1.2 Computer Simulated Physics

Another area, often associated with graphics and virtual environments, is computer simulated physics [27]. Originally conceived with the narrow aim of assisting in animation creation, physics simulations have expanded into a wide range of uses, both scientific and recreational.

Physics simulations often do not fully simulate every aspect of the real-world, instead providing a selection of acceptably realistic approximations of certain areas, such as rigid body dynamics and fluids (among others). In most cases, the aim is not to provide perfect realism, but rather to provide something that works acceptably for the user's purposes without taking unreasonable computation to achieve.

There is a wide range of real-time physics simulation software (physics engines) available, both proprietary (such as Havok[3]) and free (such as Bullet[4]). They differ slightly in their focus and areas of strength [7], but are all, at a high level, quite similar.

To summarize, computer simulated physics allows us to efficiently run simulations of real-world physics, with varying trade-offs between speed and quality, which can be used for a variety of purposes.

1.3 Problem definition

One of the problems with the procedural generation of 3D models (computer representations of 3D objects such as buildings or trees) is that the output is static. When created, the generated models will not possess any animations, movement, or support for physically realistic interaction. These components can be added by hand, but this defeats the purpose of using procedural generation. This is in contrast to the models in games and virtual environments, which are becoming increasingly dynamic and interactive (i.e. they respond to interactions, often in a manner similar to how they would in the real world) as the processing power of computers increases. Our project aims to address this deficiency of procedural generation by researching and investigating ways of extending procedural generation to automatically add physics components to generated models.

We will only examine the procedural generation of trees in this project (created using Lindenmayer systems, or L-systems). There are several reasons for this. Firstly, there is a vast and diverse range of procedural generation techniques, many of which work in significantly different ways. This means that it would be extremely time consuming to examine them all,

3 http://www.havok.com/
4 http://www.bulletphysics.com/

and very difficult to design any general techniques that could apply to all of them, due to their vast range of methods and scopes of output. Secondly, tree generation (and L-systems, in particular) is one of the more popular and commonly used types of procedural generation. Hence, this is one of the more useful areas to investigate, since it is a mature, widely used and well-understood field. Finally, it is likely that the results of this project can be generalized and extended to several other (but not all) methods of procedural generation. However, we will not look at this beyond making recommendations (based on our experiences) about how this might be done.

Although the problem of static procedural content is relevant to both real-time and pre-rendered graphics applications, we will only be looking at real-time applications. In other words, we will aim for our final graphical result to be interactive (in real-time) and operate at a respectable frame rate, meaning that it should produce at least 30 frames per second [16]. It is our belief that any method that works for real-time applications should also be trivially extensible to pre-rendered applications, but, once again, we will not investigate this.

A further motivation for this research is that procedural generation is becoming popular in video game and virtual environment development (because of its ability to reduce costs and time required) despite not supporting the creation of dynamic content.

From these two facts, we can see that it would be very useful if procedural generation techniques could be extended to create suitably dynamic models. This would allow cheaper, quicker, and easier development of computer rendered movies, virtual environments and games. The relevance and importance of this issue is further shown by the academic research that has investigated methods of making procedurally created content more reactive to it's environment [28]. This indicates an interest in, and a need for, more dynamic models.

One easy approach to making the models dynamic is to make them respond to the laws of physics (gravity, collisions, etc.). This means they could collide, break and move more realistically than if they were static, which would give a strong sense of dynamism in the environment.

It makes sense to use an existing physics engine to handle the simulation aspect of this, since there are several robust engines available that are geared towards real-time applications.

This means that the main problem of this research is devising a suitable way to integrate physics into the procedural creation process so that the end product behaves in a dynamic manner, thereby resolving the problems associated with static content.

Since we are specifically dealing with trees, we are aiming for realistic tree movements such as swaying and flexing in the wind, bending correctly when hit by objects, and falling over when uprooted.

To summarize, our main research question is:

- “Can we add the automatic creation of physics components to procedural generation techniques (focusing only on trees produced by L-systems), and if so, how?”

Some peripheral issues we will investigate are:

- How effective are the physics components in making the trees seem dynamic to users, and how realistic does the final result seem?

- What is the performance impact of adding these trees? Does it allow a real-time frame rate? If not, can we use level-of-detail techniques to achieve a real-time frame rate?

We will evaluate these questions by using the results of several experimental tests (conducted during our research) that were designed to provide conclusive answers to these research questions.

1.4 Outline

In chapter 2 we cover all background knowledge relevant to the project and examine some of the related work (in the fields of L-systems, physics and level-of-detail schemes). Following that, chapter 3 explains the methods we used to conduct our research. Chapter 4 discusses how we did our validation and user testing to evaluate our research. In chapter 5 we examine the results of the validation, and the conclusions that we drew from the research and the recommendations that follow from them are presented in chapter 6.

2 Background & Previous Work

2.1 L-Systems

L-systems (short for Lindenmayer Systems) were originally developed in 1968 by Aristid Lindenmayer as a method of simulating the growth of simple multi-cellular organisms [24]. Over time, L-systems have been expanded to be suitable for a wide range of other purposes [32]. One such use was in procedurally generating trees and other plants, the core description of which is summarized by Prusinkiewicz and Lindenmayer [39]. There is currently a wealth of research into L-systems available and the field is considered at an advanced state [39].

L-systems are simply parallel rewriting systems. They consist of an alphabet of symbols (which can be either terminal or non-terminal), a set of productions (rules which describe how a single symbol changes into one or more different symbols) and a starting symbol, or string of symbols. This may sound very similar to formal grammars, and L-systems are in fact a special category of formal grammar (with many different sub-types of L-systems), but the main difference between the two is that L-systems execute more than one production in each iteration (in parallel), whereas formal grammars only execute one. In the context of creating trees, L-systems operate on strings of characters, rewriting (and often expanding) the strings over a number of iterations.

In order to turn these strings into tree models, Prusinkiewicz proposed an approach similar to the turtle graphics used in LOGO [2,35,36]. In this approach the turtle interprets the string generated by the L-system as a series of instructions for it to follow, drawing the basic skeleton of the tree as it does the instructions. This approach is easily extended to 3D by adding extra symbols and parameters to the L-system to control the added degrees of freedom [39]. A basic 2D example is shown below to illustrate the process of turtle interpretation.

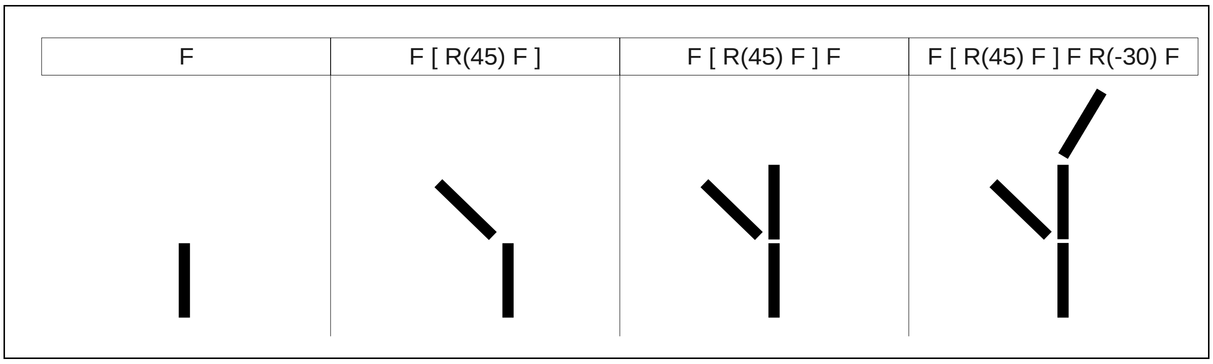


Illustration 1: An example of 2D L-system interpretation in which a simple tree is built up from a symbol string. The strings, shown at the top, are the symbol strings that correspond to the trees below them. In this L-system, the 'F' symbol is an instruction to move the turtle forwards and the R symbol an instruction to rotate the turtle's heading by the amount of degrees in the brackets.

There are a number of different types of L-systems, similar to the range of formal grammars. The simplest are DOL-systems (Deterministic, Context-free L-systems). These are both deterministic (the same inputs will always produce the same outputs) and context-free (there is no consideration of adjacent symbols when executing productions). They are simple and easy to understand, but are too simplistic and limited to be of any use in modelling trees, due to their inability to easily create branches and the similarity of successive outputs.

Next are tree OL-systems (or tree, context-free L-systems), in which a single branch segment can be replaced by a sub-tree (composed of branch segments), rooted at the base of the original branch segment. This allows tree OL-systems to form complex branching tree structures.

A more complex type of L-system are bracketed OL-systems. Invented by Prusinkiewicz [35,36], these introduce the notion of a stack into the interpretation of the string, with brackets used to indicate the pushing and popping of the turtle's current state (position, direction, etc.). This introduction of a stack to store the turtle's state allows the easy creation of branches and branching structures in the generated tree (to make a branch simply push the turtle's state, do the instructions for the branch and pop the stack to return to where the branch started from).

A further extension to L-systems are stochastic L-systems [13,49]. These L-systems were developed to compensate for the fact that deterministic L-systems will always generate the same output. In a stochastic L-system each production has a probability of occurring, and there is generally more than one possible production for each situation, which means that each time a production is executed, the output is randomly chosen from a number of options, based on the probabilities associated with those options. Due to this, generated trees will all be different (even for the same input), and hence the algorithm is non-deterministic.

All of the preceding types of L-systems have been context independent, that is, any production could occur anywhere in the generated tree. Context-sensitive L-systems change this by constraining where a production may occur. There are a number of subtypes [18,26,43], but they all share the characteristic that certain productions will only be executed if the symbol(s) before and/or after the current symbol (the one the production is being used on) match the symbols(s) specified in the context-sensitive production. Using these L-systems it is possible to gain more control over the generated content and simulate plant growth [39].

Finally, there are parametric L-systems. Despite the range of output possible in the previous L-systems, they are still fundamentally limited in a number of ways. For example, all lengths in the plant will be restricted to multiples of whatever the unit length of the system is. Parametric L-systems were created to counter this deficiency and were first suggested by Lindenmayer [25]. In parametric L-systems the characters in the string that the L-system works on, and updates, may have associated parameters (generally real numbers, but in theory they could be anything). These parameters can represent information and attributes about the symbol, such as branch length or trunk thickness, and they may be modified by productions for the new characters that are produced from the old character (or the parameters can be directly inherited through productions). Parametric L-systems allow a much greater degree of variation in output than other types of L-systems, and are important in producing trees that look realistic.

A slightly newer extension to L-systems is environment-sensitivity [17,28,38]. There are several ways of doing this, but in essence they all involve the generated plant model being adjusted in accordance with environmental factors. Some examples are: using positional information to control the direction of growth of the generated plants so that the end result appears similar to topiary [38], using environmental light to influence how a plant develops over time [17], and having adjacent trees adjust themselves to compete for the available light [28]. The communication can be bidirectional between the plants and their environment, creating a two-way feedback system in which the plants and the environment can both affect each other (as is done in open L-systems [28]). Alternatively, the communication can be one-way, meaning that the environment can affect the plant, but not the other way around (for example, environmentally-sensitive L-systems [38]). One example of how this environmental-awareness is done are open L-systems, which introduce query symbols into the symbol strings. These query symbols are evaluated, based on the environment's state, when the tree is interpreted and the results of the query influence the execution of productions while the L-system is running.

L-systems have also been used to model and visualize the progress of plants growing over time. One such approach for doing this, proposed by Prusinkiewicz et al., uses a new type of L-system, differential L-systems. This type of L-system introduces the idea of a continuous flow of time, instead of discrete iterations of the L-system's derivations [37].

Additionally, these two approaches have been combined, allowing the simulation of plant growth affected by the environment. Once again approaches vary, especially in the realism of the model used. For example, Van Haevre et al. used a ray density technique to estimate light levels and their effect on plant's development [17], while Lam and King's approach involved a model much closer to how real plants develop (buds, water availability, etc.) [23].

It is important to note that none of the above variations of L-systems are mutually exclusive. For example it is possible for an L-system to be parametric, context-sensitive and stochastic. In practice, most extensions to L-systems can work together, combining their strengths and allowing the generation of very realistic trees. In fact, most uses of L-systems take this approach and combine the different types of L-systems to maximize the generational power of the application.

Finally, L-systems have been combined with plant distribution models and procedural landscape generation techniques to create entirely procedurally generated scenes. These are often nature based (forests, jungles, etc.) [11], but it has also been done with large scale cities [32]. These entirely procedurally created environments show the ability of L-systems to create content for games and virtual environments.

2.2 Physics

Computer simulated physics was first introduced during the late nineteen eighties as a way to lighten the workload of animators by automating the animation of certain objects in virtual environments [27].

Since then, however, the field of computer simulated physics has grown significantly, with the technology being used in many areas, such as computer games, movies, scientific experiments and more. It is also still an active area of research, and there is interest from several industries in improving physics simulations and their efficiency. There are a number of mature software packages which do computer simulated physics (commonly known as physics engines) available[5] [6] [7].

The techniques have also become more sophisticated. Although physics engines did and still do use Newtonian physics to simulate events, originally very simple models (for example, particles and mass-spring systems) were used. These are significantly restricted compared to the more advanced and powerful engines that are in use today (which support features such as fluids and soft-body dynamics) [27].

The biggest distinction to make in computer physics systems and their applications is that of real-time versus off-line physics [27]. Off-line physics are typically used when the accuracy of the simulation is of key importance and real-time performance is not required, for example, in scientific experimentation or movie special effects. In these systems the engine focuses on being as correct and accurate and possible, at the expense of speed.

On the other hand are the real-time physics simulations. In these, maintaining an acceptable frame rate (at least 30 frames per second [16]) is the most important requirement. Simplifications and optimizations at the expense of accuracy can be used to speed up the process and, as long as everything looks acceptably correct (where the definition of 'acceptably' depends on the specific application), the quality of the simulation can be allowed to degrade in favour of performance. These physics simulations are most commonly used in video games, virtual environments and other areas where real-time visual feedback is necessary.

An important development in computer physics is the idea of general-purpose computation on GPUs (GPGPU). This idea refers to using the graphics processing unit (GPU) found on modern graphics cards to do non-graphics related computation. Current graphics cards have introduced functionality to program the rendering pipeline with arbitrary instructions using programs called shaders, and general purpose frameworks which allow access to the GPU's resources for non-graphics processing, such as CUDA[8]. This does not mean that any program can be run on the graphics card though. Due to the type of data graphics card are optimized to work on, only programs which are stream-based can be run on the GPU [51].

Luckily computer physics are well suited for running on the GPU, and significant performance gains can be made by taking physics calculations off the CPU, and onto the GPU [20]. This means that doing physics simulations on GPUs is a growing area for performance gains and research. A good example of this is Nvidia's PhysX engine[9] which has extensive GPU support.

As mentioned previously, physics can also be used to procedurally generate animations. These techniques generally operate by creating a simplified physics skeleton of the entity to be animated and applying physics simulations to that skeleton to produce animation without explicit input from a human [15,34]. One of the best known examples of this are the 'ragdoll' physics found most commonly in games (for example, Unreal Tournament 2003[10]), where characters 'realistically' fall, roll and collide with their environment. This approach can be useful, and save animators large amounts of work, but the problem with this approach is that, without designing some sort of controlling system, it can be difficult to control the generated

5 http://www.ode.org/
6 http://developer.nvidia.com/object/physx.html
7 http://www.bulletphysics.com/
8 http://www.nvidia.com/object/cuda_what_is.html
9 http://www.nvidia.com/object/nvidia_physx.html
10 http://www.unrealtournament2003.com/

animations and what they will look like [34], which can be a problem for certain applications.

In the area of animating trees specifically, the dominant approach seems to be that suggested by Sakaguchi et al. [42] and Wong et al. [48]. The approach is quite similar to ragdoll physics used in character animations. In this approach the tree's branches (including the trunk) are broken up into circular or capsular segments of varying length with defined physical constraints between the adjacent segments (appropriate joints, such as ball and socket joints, with a high rigidity). Each branch uses relative coordinates so that when its parent branch moves the child branch moves with it. The physical simulation is then applied to the segments so that when external forces act on the tree it moves in a realistic and convincing manner. This approach can also be extended to the leaves of the tree [48].

Another useful area of research is comparisons and evaluations of the various freely available physics engines [7,45]. Work such as this gives indications of which engines work best for which criteria, and other useful advice, such as the the use of an abstraction layer (designed to work with multiple engines without any work on the user's part) when utilizing a physics engine in a program [7]. However, the authors note that a generalized abstraction layer may restrict access to unique or specialized components of the various physics engines. Although most of the features evaluated and compared in this research are more advanced than necessary for our goal of adding physics to procedurally generated trees, the papers we reviewed that compared physics engines were very useful in planning our implementation.

2.3 Level Of Detail

In the context of computer graphics, using a level of detail (LOD) scheme refers to lowering the detail of objects as they move further away from the viewer. Due to the increased distance between the object and the viewer, the drop in visual quality is not noticed by the viewer and the lower detail of the objects means that less processing power is spent rendering them [8]. This increases performance and is very important for large outdoor scenes where lots of geometry is to be displayed (for example, forests).

The first distinction to make is discrete versus continuous LOD schemes [50]. In discrete schemes each displayable object has a finite number of models of varying detail. Each of these models is assigned to a certain distance range, and when the viewer is within a certain distance range, the corresponding model for that range is displayed. This approach requires more initial work when creating the different models and may cause 'popping' artefacts (where the change between different levels of detail is visibly, and distractingly, noticeable to the user), but it requires less computation during the actual simulation/real-time display.

In continuous schemes, the polygons on models are adjusted in real-time to produce a model that provides an acceptable balance of visual quality and performance, taking the distance to the viewer into account. This approach negates the problem of 'popping' and only requires one full detail model to work with (instead of one per distance range), but it needs more computation while the simulation/game is running, which may degrade performance.

There are also schemes that combine continuous and discrete approaches, such as Zach et al's point based rendering approach [50].

Most of the literature focuses on complex algorithms to do the adjustments to the models (either continuously or for the discrete steps). However, this is of little use to us since procedurally generated plants are described by a string of symbols and not a 3D model. Hence these simplification algorithms are of limited use for our purposes, and comparable benefit (for much less computation) can be gained by simply reducing the number of polygons in the component shapes of the generated trees in proportion to the distance between the camera and the tree.

When it comes to trees, specifically, a very common approach in LOD schemes is billboarding [6]. In this scheme, distant trees are rendered as 2D images oriented to face the viewer instead of 3D models. This change is not visibly noticeable for large distances and can provide significant performance gains, which makes it a good choice. Billboarding is generally regarded as the solution to LOD for trees, as can be seen by the large number of real-time games that use it[11]. However, a problem with billboarding is that the billboarded trees are static (essentially images) and we are attempting to create dynamic animated trees.

Another approach for level of detail scaling when simulating trees is described by Beaudoin and Keyser [5]. They suggest simplifying multiple branches into a single 'average' branch in such a way that the average branch approximates the original branches. This process can be repeated on the average branches as the distance to the viewer increases and detail is allowed to drop.

2.4 Previous Work

The idea of using physics to simulate trees is not an entirely new one. There has been a smattering of research that has trod similar ground. Here we briefly describe the two papers of most relevance to our investigation.

Firstly is Sakaguchi and Ohya's work on modeling and animating trees [42]. They were working on animating trees for use in virtual environments and shared some of our intended outcomes, however they generated their trees from captured image data instead of L-systems. In spite of this, parts of their research inspired the method we used in our system, notably their division of the tree model into rigid segments which are simulated individually to produce realistic behaviour in the overall tree structure. Our research goals differ from theirs though, in that they wanted to use a highly realistic physics model and real-time interaction was not a priority in their system.

Additionally, Wong et al. did some similar work on animating plants [48]. Their technique was intended to be usable in real-time situations and followed a similar approach to Sakaguchi's by dividing the plant into rigid segments and placing constraints on the movement between adjacent segments. Their research was aimed at animating low numbers of small soft plants though, and placed a heavy emphasis on leaf animation. In contrast we are more interested in large numbers of fully grown trees. As with Sakaguchi though, their research gave useful insight into the problem and helped us develop the approach we investigated in our research.

11 http://www.speedtree.com/gallery/

3 Design and Implementation

3.1 Introduction

In this section we discuss the design and implementation details of the software created for this research.

In order to test our research hypotheses it was necessary that we create a prototype system. We decided early on that this would not be production-grade software; all that was needed was something that would serve as a proof of concept. This allowed us more leeway in creating the software, and saved a lot of time that would have otherwise been spent on re-factoring, maintainability and intense optimization.

Essentially what we needed was a simple games engine with a robust physics engine. Visual quality was not a major requirement, since the focus of the project was on the movement and dynamics of trees, not their appearance. The physics, however, as a major aspect of the research, needed be as realistic and accurate as possible.

3.2 Technical Details

Since real-time performance was an important concern we decided to program in C++. C++ provides the benefits of being a high performance compiled language, with object orientation. Since we were aiming for a high level of modularity, we opted for an object oriented language. Our intended platform for the system was Linux/Unix, and all programming was done on Ubuntu Linux using an iterative development model.

From the start of the project we were under time constraints for completion, and hence we opted to use as many pre-existing components as possible in our program. This shortened the development time of the experimental framework and allow us to begin experimentation sooner.

We found that the best option in pre-existing components was stable, mature software libraries that were under free-to-use (as in no money was required to use them), or open source licenses. After some investigation and consideration, we decided to use OGRE[12] as the graphics engine, and PhysX[13] as the physics engine. Our rationale was that both of these are proven software that have seen use in the games industry, have been under continuous development for a very long period (9 and 5 years respectively), and are still maintained.

We also used Qt[14] (a GUI and toolkit library) to create a GUI and handle input for our software, and CMake[15] to handle automated compilation.

The framework that we used in this report was shared with another honours project, investigating the addition of physics components to procedurally generated buildings. Most of

12 http://www.ogre3d.org/
13 http://developer.nvidia.com/object/physx.html
14 http://qt.nokia.com/
15 http://www.cmake.org/

the code was common to both the projects, and the differing parts were kept isolated to prevent them from interfering with each other. A diagram of the high level structure of our framework is included below for reference.

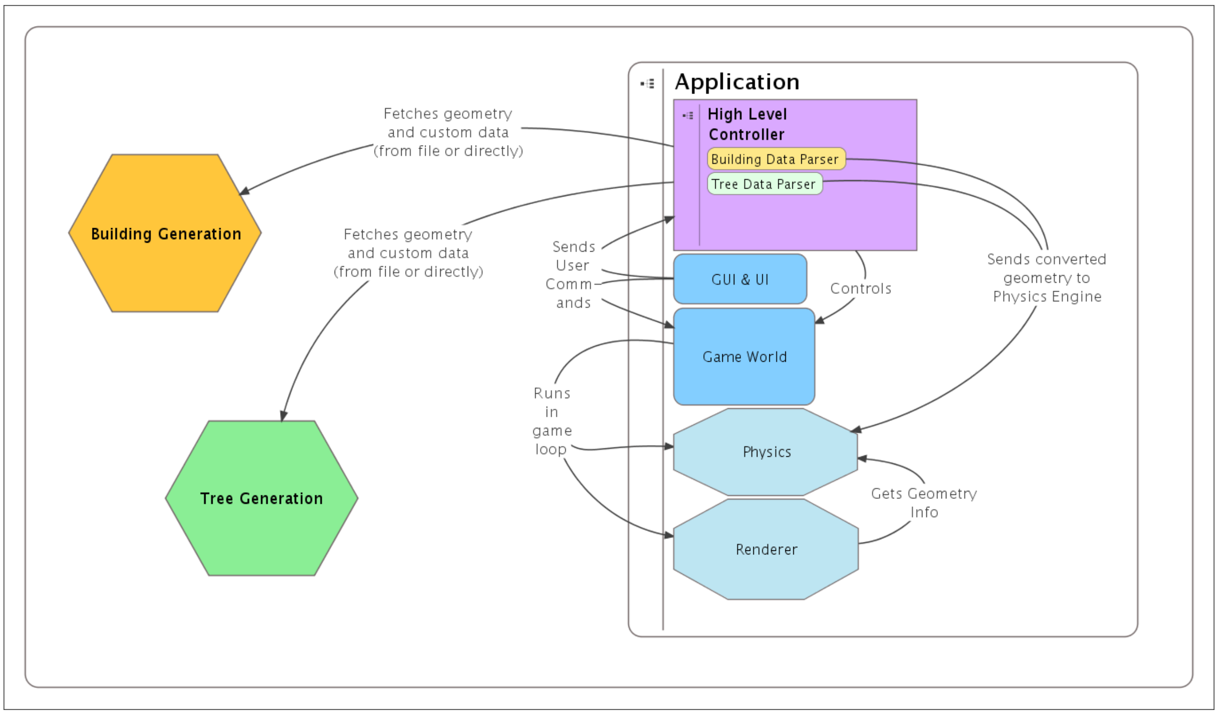


Illustration 2: A diagram of the high level structure of the framework we developed in this research. The framework was shared with another similar research project, and includes components that were not relevant to this research (such as Building Generation). They have been left in this diagram for the sake of completeness and full disclosure.

3.3 L-System Parser

One of the first and most important components we created was an L-system parser that we could use to process L-systems and produce sets of symbols to create trees from.

We initially considered using a pre-existing L-system library to handle this part of the project, but, after some investigation, decided against it. Open source L-system libraries are scarce, and we were advised that we would probably encounter limitations in the packages that were available. For these reasons, we opted to create our own L-system parser instead.

We decided to make the parser a standalone program, completely separate from the main program used to display and run the physics simulations. This was mainly due to the simplicity and abstraction that this brought. With the programs separate, they would both be that much simpler, making it easier to debug and improve them. This also meant that any major changes in either software component would not have a ripple effect on the other.

The L-system parser was also written in C++, and used the flex++ lexical analyser[16] to assist

16 http://flex.sourceforge.net/

in reading and parsing the input L-system. Our L-system syntax was simple enough that our parser used a very basic LR(1) parsing paradigm.

The program was command-line based and operated by reading in an L-system grammar, parsing it, and using it to produce output strings of symbols in accordance with parameters specified by the user. The input and output were both done with plain text files for maximum compatibility and simplicity.

The parser supported several of the enhancements to L-systems, and could use them all in combination if required. These enhancements were chosen based on those that were most useful and applicable to our context: parametric, bracketed, context-sensitive and stochastic L-systems [39].

There were also the parameters we used for controlling the output from the parser. The most important one was the ability to control the maximum number of iterations. Since not all L-systems naturally run to a state with only terminal symbols, this was important to ensure that the program did not run forever. We also had support for creating multiple separate strings from the same L-system (to generate many different instances of a type of tree for example).

3.4 Tree Creation

Once the L-system parser was working correctly and we had verified it against some classic test examples found in the literature (namely simple 2D fractals and plants found in the first few chapters of The Algorithmic Beauty of Plants [39]), we were able to move on to generating trees from the symbol strings that it produced.

By this stage, the framework was at mostly functional (graphics, physics simulation and controls all operational), which meant that we could visually observe the 3D results of our tree creation algorithms. This was extremely important in verifying that we were on the right track. The process of creating the trees (without physics support) is very similar to the turtle graphics approached proposed by Prusinkiewicz [39]. The two main differences are that we initially stored the tree as a collection of points instead of a collection of lines (see illustration 6), and that we use a hierarchical tree data structure to store the details of the tree (see illustration 4), instead of only storing the symbol string associated with the tree.

The conversion from a sequential symbol string to a hierarchical tree structure is non-trivial. Our algorithm operates recursively by reading in symbols from the input string and performing the corresponding turtle interpretation (move forward, rotate about an axis, etc.) to build up a sequence of points that represent a branch in the tree. When a '[' symbol (indicating a branch where the current state should be pushed onto the stack) is encountered, the algorithm recursively calls itself on the string encapsulated between the encountered '[' symbol and its matching ']' symbol (indicating the end of a branch, and that the state stack should be popped). This string, contained between the two bracket symbols, represents the instructions to create the child branch.

Once the algorithm has reached the end of a symbol string, it returns an ordered list of 3D points, the attributes that are associated with those points (such as branch thickness, texture, etc.) and any child nodes that the branch may have. This list of points is added to the tree data structure as a child of the parent node (the main trunk is the root element of the tree data structure and does not have a parent).

Pseudocode of our algorithm is found in illustration 3.

It should also be noted that each child branch is associated with a point on its parent branch; the position at which the child connects to its parent branch. There is no limit on the number of children that can be associated with a single point on the parent.

To see a visual example of the final hierarchical data structure, please see illustration 4, which shows the final result of the conversion algorithm for a simple example.

```
buildTree(symbolstring)
    tree <- empty TreeNode;
    loop i through symbolstring
        if i is '['
            s <= all symbols from i+1 to the corresponding ']' symbol;
            tree.addChild( buildTree(s) );
        else
            if i is 'F'
                add a segment to tree;
            else
                update tree based on what type of symbol i is;
            endif
        endif
    endloop
    return tree;
```

Illustration 3: Pseudocode of the recursive algorithm used to build a hierarchical tree data structure from the symbol string created by an L-system.

Some technical points to note about this process are that any unrecognised symbols are ignored (which also allows the addition of comments to the tree descriptions) and all rotations are implemented using quaternions, to avoid the gimbal lock problems of Euler angles [1].

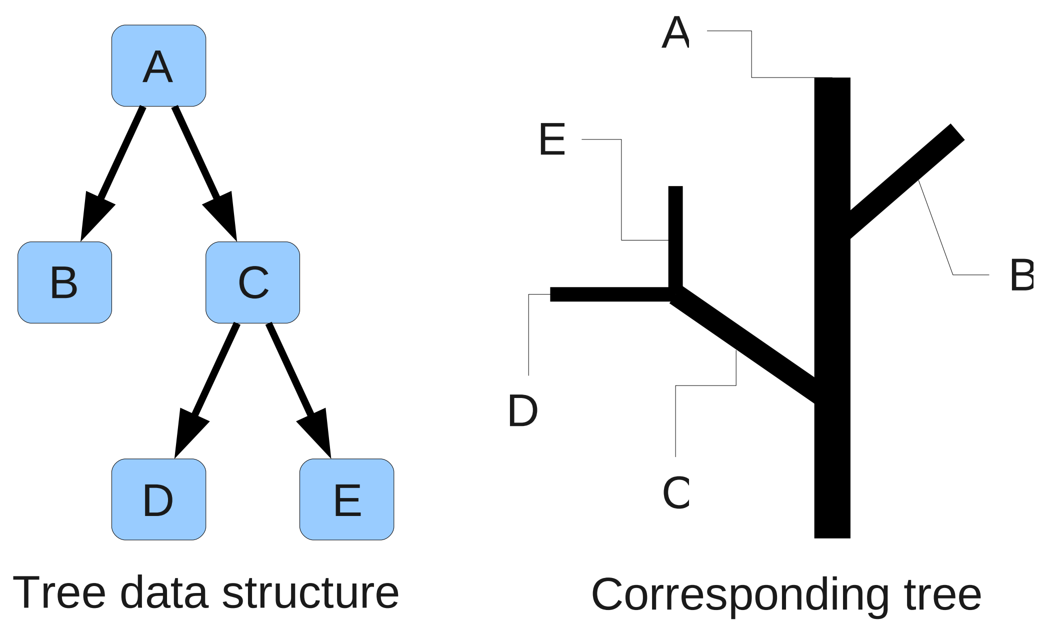


Illustration 4: This shows the relationship between the hierarchical tree data structure that stores a tree and the actual tree itself. Each tree limb (chain of segments) is stored in a node, with children of that node being the branches that come off the limb.

3.5 Physics Skeleton Creation

After creating an abstract tree of points, the next step is to express the tree in a representation usable by the physics engine to simulate the movements and interactions of the tree.

Our method of doing this is to replace each segment (the lines formed between successive points) of the tree branch with a rigid physics 'bone' and connect the chain of bones together with appropriate joints. This is based on the approach used by Sakaguchi et al. [42], but is modified for adaptation to using a physics engine. Please see illustration 6 for an example of our algorithm in practice.

In our implementation, the rigid bones are represented using capsules (essentially 3D ellipsoids). This was not our first choice. We had intended to use cylinders as they approximate branch segments better. However, the physics engine (PhysX) did not have support for cylinders and capsules were the next best option. We recommend that anyone implementing our algorithm make use of cylinders, due to their closer similarity to tree branches.

Our algorithm operates recursively on the point-list data structure created in the previous step. It iterates along a branch, creating a capsule between each pair of points in the branch according to the physics parameters associated with those points (weight, branch width, etc.). These parameters are obtained from the L-system symbols. When it encounters a point with associated child nodes, it recurses on those, using the point from the parent as the starting point in the child branch. Pseudocode is included, in illustration 5.

```
makeSkeleton(rootnode)
    loop i through branch segments in rootnode
        p <= new physics capsule made from physics parameters in i;
        create p between the points at i and i+1;
        create constraints between p and the segment before it;
        if i has children
            loop j through children
                makeSkeleton(j);
            endloop
        endif
    endloop
```

Illustration 5: Pseudocode of the recursive algorithm that is used to create the physics skeleton of a tree from the hierarchical tree data structure that stores it.

The visual representation of each branch segment is also created at this stage. Each branch segment is represented by a textured cylinder. Since a cylinder does not perfectly fit within a capsule, this can cause problems during collisions because the visual representation may not match the underlying shape used for collisions. In practice, we found that for reasonable dimensions of the cylinders and capsules this was not a problem. Still, this can be avoided by using cylinders for the physics representation (as discussed above).

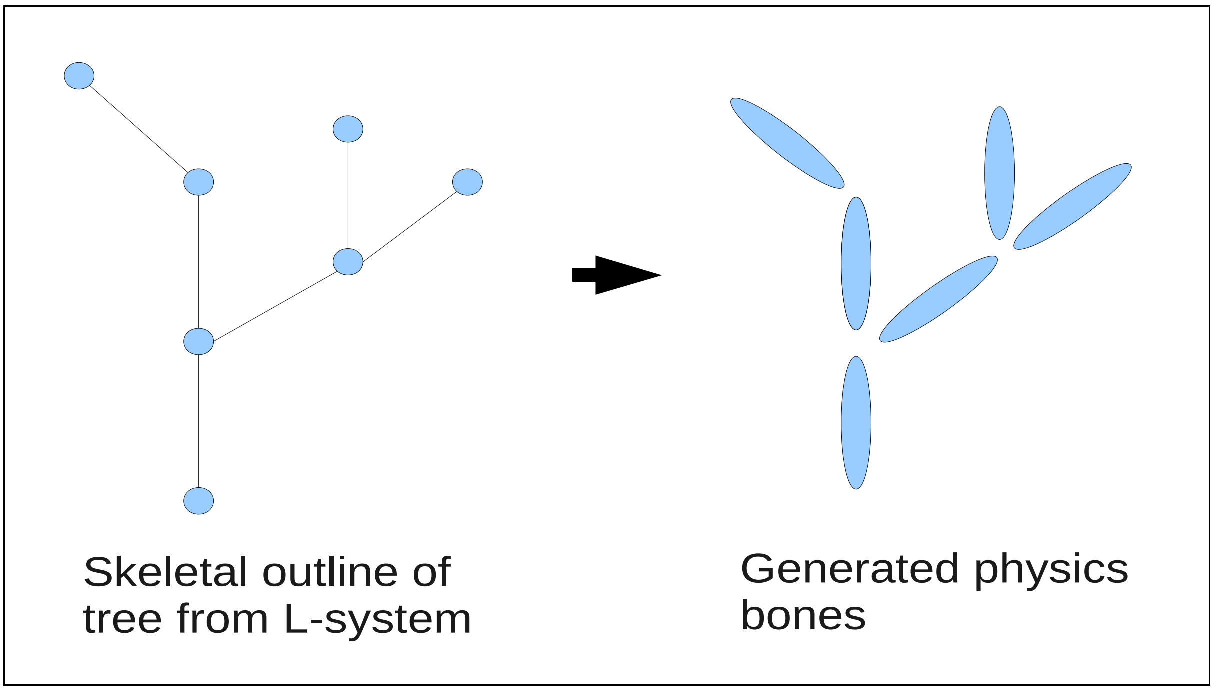


Illustration 6: The final result of our algorithm to create the physics bones for the tree. Along each segment a rigid bone is created that is slightly shorter than the segment itself. These segments are later connected using constraints. The bones are not, themselves, rendered. Instead a cylinder is rendered in their place.

In practice we found that it was necessary to make the capsule slightly shorter than the actual distance between points. If it spans the full length, the ends of adjacent capsules overlap which causes undesirable movement, interacts badly with the joints and a number of other problematic issues. For this reason it is strongly recommend that the physics bones be made between 80% and 95% of the actual length of a branch segment, as shown in illustration 7.

The fact that the visual representation is separate from the physics representation also means that the visual cylinder can be specified to the full length of the branch segment. Without this, there would be obvious visual discontinuities between segments.

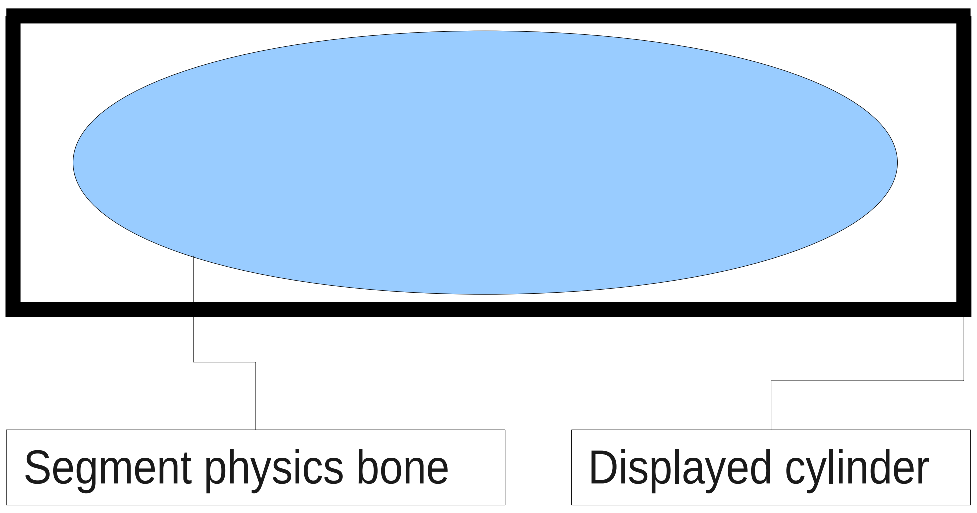


Illustration 7: A diagram showing how the physics bone is created to be shorter than the cylinder used to visually represent it. The displayed cylinder extends the full length of the branch segment and is what is visually rendered. The actual physics bone itself is only some fraction of the segment length (0.8 to 0.95 in practice).

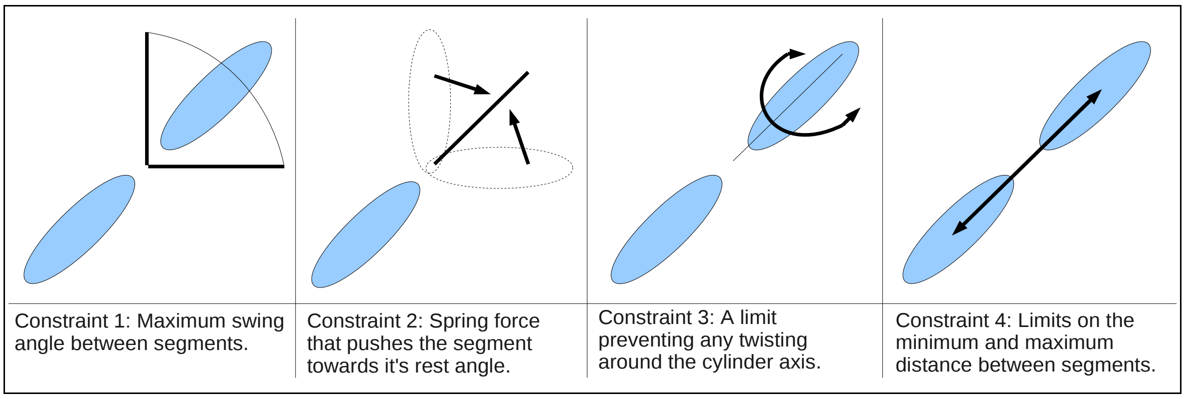
Once the basic bones and their visual representations have been created, it is necessary to specify how bones are connected. If this is not done, then the trees will simply fall apart under the force of gravity.

Our method of connecting the bones is to join each bone to those before and after it, as is typical in character animation. Since the final segment in a branch has no successor, it is a terminating bone. Also, the first segment in a branch will connect to its parent branch (or the ground in the case of the main trunk).

There are four constraints in each joint (see illustration 8):

- A spherical joint designed to control the angle between two adjacent segments (1 in the diagram). This prevents movement past a specific limit.
- A spring force that moves the segments back to their rest positions (2 in the diagram). The spring is important in giving the branches a sense of springiness as seen in many plants.

- A constraint that prevents the segments from twisting around the axis running down the length of the cylinder/capsule (3 in the diagram). Since the branches on real trees do not show much, if any, rotation of this type, this is important in maintaining realism.
- A distance constraint that limits the distance allowed between the joined segments (4 in the diagram). This means that a minimum and maximum distance between the segments can be enforced. This is added to prevent strong forces from overcoming the spherical joint's constraints and moving adjacent segments apart, creating a visible discontinuity.


Constraint 1: Maximum swing angle between segments.
Constraint 2: Spring force that pushes the segment towards it's rest angle.
Constraint 3: A limit preventing any twisting around the cylinder axis.
Constraint 4: Limits on the minimum and maximum distance between segments.
Illustration 8: The three types of constraints in a joint between two branch segments. Although they are represented in two dimensions here, those used in our system all work in three dimensional space. Additionally, in our system, all parameters for these constraints can be varied on a per-joint basis using symbols in the L-system.

Each of these joints also has an optional breaking force which, if exceeded, will cause the joint to be destroyed and no longer act on the two segments. This allows the trees to break in a realistic manner.

Additionally, all of the parameters associated with these joints are adjustable on a per segment, or global, basis via symbols in the L-system. This is important in allowing the creation of the many varied types of trees found in nature, since different species often have very different rigidities, breaking strengths, branch springiness, etc.

Although different physics engines support different types of joints, any reasonably functional engine should support the constraints used here, allowing our scheme to be implemented under any physics engine.

3.6 Optimizations and Level of Detail Schemes

There are a number of performance-improving enhancements included in the final system. These are aimed at improving real-time performance, and can be roughly divided into two categories: those that operate on the visual, graphics part of the program, and those that operate on the physics subsystem.

The graphical optimizations are all established techniques and are well understood. Two of these were provided by the graphics engine (OGRE) and required little to no work for us to incorporate. As these techniques are not the focus of this research, we will not describe their operation here.

The graphics optimizations are:

- Frustum visibility culling [4] (integrated into OGRE)
- Backface culling [22] (provided by OGRE)
- Distance threshold culling. Under this optimization, any object further than a certain distance from the camera (a variable threshold) is not rendered. The lists of objects to render is updated twice a second (a heuristically chosen value designed to balance visual appearance with performance). This is implemented as a very rough approximation of tree billboarding [6]. This approximation does not take the cost of drawing the billboards into account, but, since the relative complexity of the billboards is much lower than actual trees, the time to draw the billboards is small and this is a (relatively) minor discrepancy. This, combined with the fact that implementing a fully-featured billboarding system is beyond the scope of this project, allows an acceptable approximation to billboarding for the purposes of this research.

The optimizations for the physics side of the program are simply graphics optimizations adapted for use in the context of a real-time physics simulation. The physics engine also provides some optimizations, the major one of which is the ability to put objects to 'sleep'. When this is done, the sleeping object is not simulated unless some external force acts on it and 'wakes' it. PhysX also does some internal automatic sleeping on objects that have not moved significantly for some period of time.

The physics optimizations that we implemented are as follows:

- Frustum culling. This is similar to frustum culling in a computer graphics situation. Any object that lies outside a slightly widened version of the camera's viewing frustum does not need to be simulated (slightly widened to prevent objects spanning the edge of the screen from being affected). This scheme is not perfect and can cause some inconsistencies with what the user would expect. For example, consider an object being thrown into the air, with the camera not observing it for some time, and then returning later. The object will still be in the air when it should have fallen and landed on the ground some time ago. Nonetheless, for certain situations, such as trees moving in a light wind, this optimization can work well.
- Distance culling. An equivalent to billboarding in a graphics context. Any object beyond a certain range is put to sleep, and not simulated. Because the object is some distance away, the fact that it is not being simulated is not obvious to the viewer. This is not a perfect solution (situations similar to the one described above can occur), but, like frustum culling, it can work well under certain circumstances. Unlike the graphical distance culling, this distance culling is only updated once per second (as its impact is less visually striking).

It should be noted that both the physics, and graphics distance culling used the same distance threshold. The value of this threshold can have a large impact on performance and visual quality and therefore needed to be determined with some thought. The threshold was determined heuristically during development, by viewing many different types of trees at a range of distances and deciding on a distance at which we felt a billboard would provide an acceptable substitution for the actual tree model.

This value was also used as the physics threshold. The reason for this was that making the physics threshold larger would result in wasted processing on simulating objects that would not be drawn, and making it smaller would result in unmoving trees at the edge of the threshold which would appear unrealistic, given that closer trees would be moving.

We admit that this approach lacks rigour and is not the best method for deciding on this parameter. However, we struggled to find a satisfactory algorithm to determine the optimal distance, and due to time constraints, had to resort to this approach. Additionally, we emphasize that the main focus of this research was on the creation of the physics components, not on the level-of-detail schemes.

In the end, the threshold distance was given a value of 600 units. To give an idea of scale, the tree size ranged between 20 and 80 units wide (including the canopy), and between 60 and 150 units tall.

In the interests of research configurability, we made all of the optimizations as customizable as possible. This was to ensure that we would not be limited in the experimentation section of the project.

For an investigation into how important these optimizations are, and how well they work, please see section 5.2 of this report.

3.7 Leaves

In order for our trees to appear more realistic we opted to include leaves. This is intended to be a very basic system and its main goal is to ensure that, during the user testing, users would not be biased against the appearance of the trees.

The system works by defining a symbol in the L-system to indicate whether a branch should have leaves on it, and another symbol to specify the type of leaf to put on that branch. The leaves consist of two rectangular billboards textured with a leaf image (see illustration 9). These billboards are placed perpendicular to each other, intersecting the segment such that large portions extend out on either side of the branch segment. The visible extruded part gives the appearance of leaves, based on its texture. In practice, we used alpha mapped textures with many leaves detailed per texture, giving the illusion of multiple leaves per branch segment.

These leaves do not have any associated physics, and hence the only physics simulation they undergo arises from being attached to a branch segment and moving with that segment.

We did some preliminary investigation into animating the leaf textures, or otherwise finding a cheap method of making the leaves seem more dynamic, but time constraints forced us to abandon this.

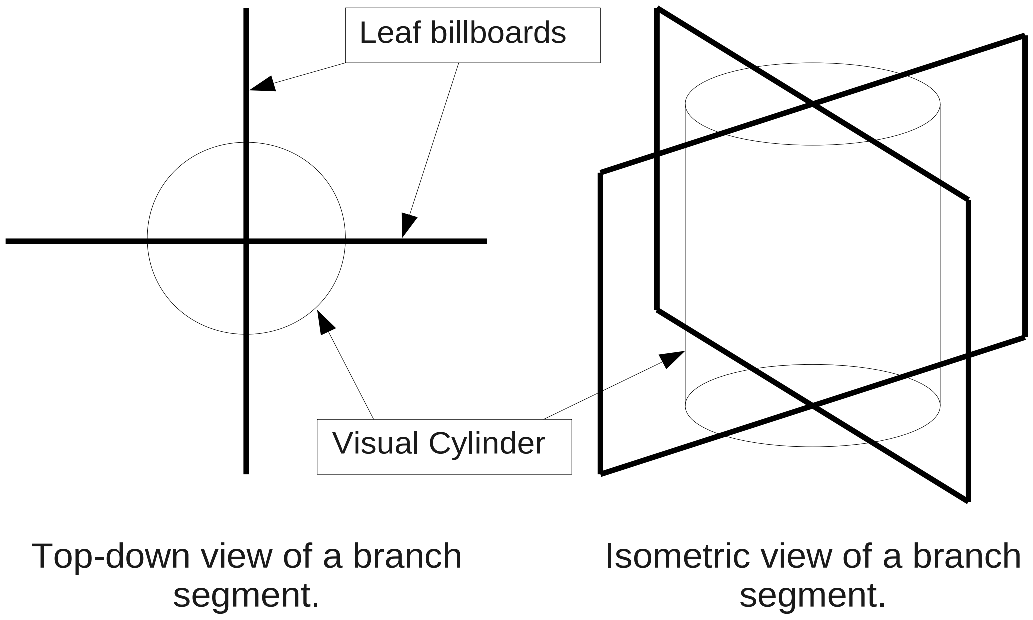

Leaf billboards
Visual Cylinder
Top-down view of a branch segment.
Isometric view of a branch segment.

Illustration 9: A top down, and isometric view of a segment cylinder with attached leaves, showing how the leaf billboards are connected to the cylinder. The billboards are always perpendicular to each other to prevent any angles at which the leaves might not be seen (excluding viewing from directly above the cylinder).

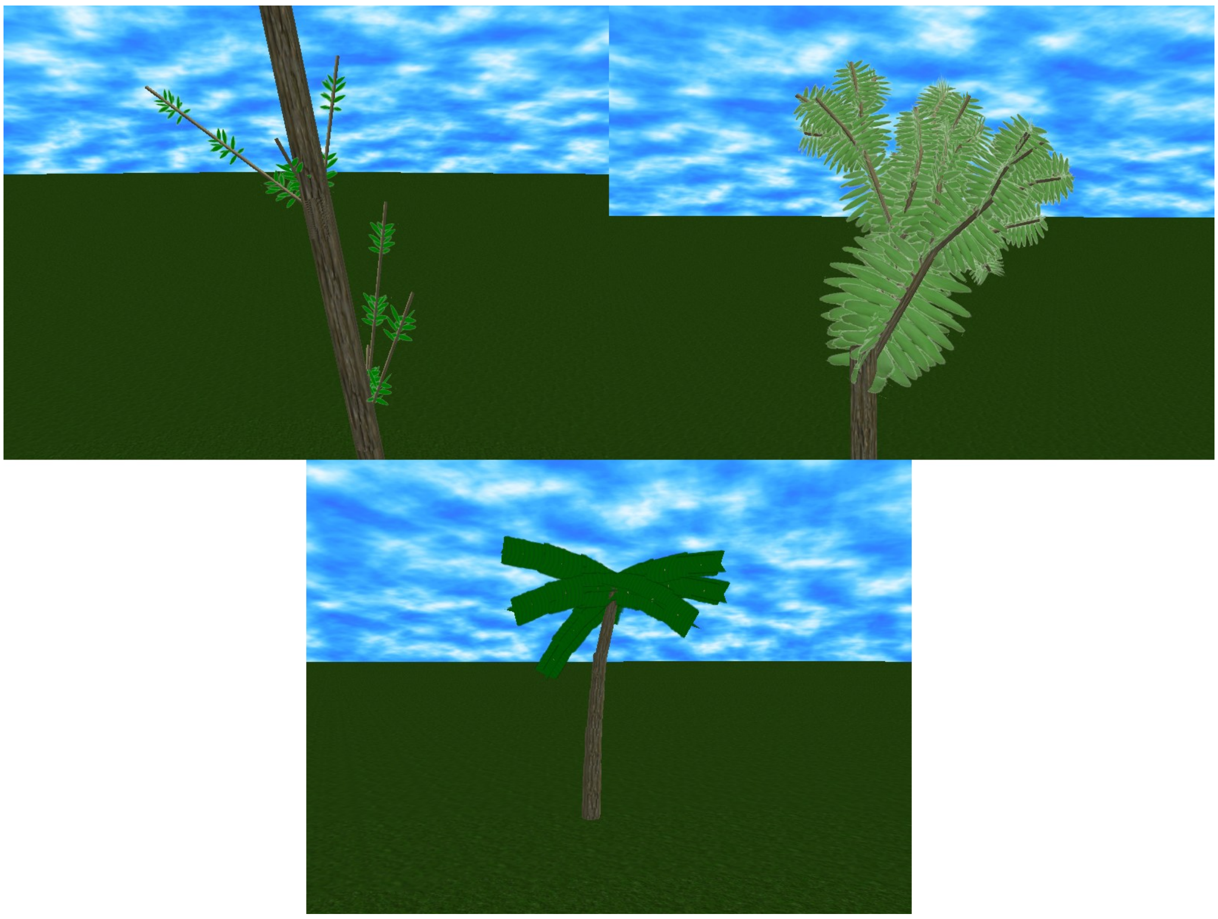


Illustration 10: Some examples of our leaf scheme as seen in practice on three different varieties of trees.

4 Experimental Design

4.1 Overview

Returning to our original research questions, the two aspects that we needed to investigate were as follows (summarized from the introduction section):

- How does the system perform? Can it achieve a real-time frame rate?
- How realistic and convincing do the movements and dynamics of the generated trees seem?

In order to answer these questions we chose three different sets of experiments:

- A thorough set of performance tests measuring the frame rate of the system under assorted conditions.
- User testing to determine how realistic the trees seem to the 'average person'.
- A rigorous analysis of the trees' behaviour, aimed at determining how realistic the trees are when compared to actual trees.

The first set of tests is intended to answer the first question (regarding the system's real-time performance), and the second two test sets are aimed at the second question (the realism of the created trees). Since none of these experiments rely on each other, we chose to run them independently.

4.2 Performance Testing

In order to evaluate how well our system performs computationally, to what extent it is capable of running in real-time and how important culling and optimizations are, it was necessary to run a suite of tests on the system, under varying parameters.

4.2.1 Variables

The independent variables in these tests were:

- The number of simulated trees in the scene.
- Whether distance culling (for both graphics and physics) was enabled.
- Whether backface culling was enabled.
- Whether frustum culling of the physics was enabled (visual frustum culling was always enabled, see below).

The dependent variable was:

- The time taken to perform updates of the simulation.

4.2.2 Test Procedure

Our test approach involved establishing a base case for the simulation, that is, the standard configuration of the system. We then ran a suite of tests, where, in each test, only one variable was modified from the base state. In this way, we could test the effect of each modification in isolation from the others. It was important that only one variable be modified in each case, to prevent the effects of multiple variable modifications from interacting with each other and producing misleading results.

For our testing, the base case was the system with all culling schemes enabled (backface, frustum and distance).

The testing cases then consisted of the following situations:

- All culling schemes enabled except backface culling.
- All culling scheme enabled except physics frustum culling.
- All culling schemes enabled except distance culling.
- All culling schemes disabled.

The base case, and all of the testing cases were tested with 0, 50, 100, 150, 200 and 250 trees in the scene (with 12 runs being done for each number of trees).These specific numbers of trees were chosen because:

- They represented a wide range of simulation scenarios and three orders of magnitude.
- With billboarding or distance culling enabled, it is unlikely more than 250 trees would need to be simulated at once.
- This is a linear progression which is useful for plotting graphs and determining the rate of growth of update time as a function of the number of trees.

Each run consisted of the camera moving on a fixed path (see illustration 11) through a forest of trees (the path was common across all testing cases to prevent unwanted factors from affecting the outcome). The path started outside the forest, moved into it's centre, performed a full revolution, and moved out the opposite side of the forest. This specific path was chosen because it is a way to evaluate frustum culling, since it covers all the types of scenarios for frustum culling, from worst to best. The trees were evenly distributed inside a fixed space (regardless of the number), so the number of trees for the testing case was proportional to the forest's density. The performance data was recorded while the camera was moving along the path.

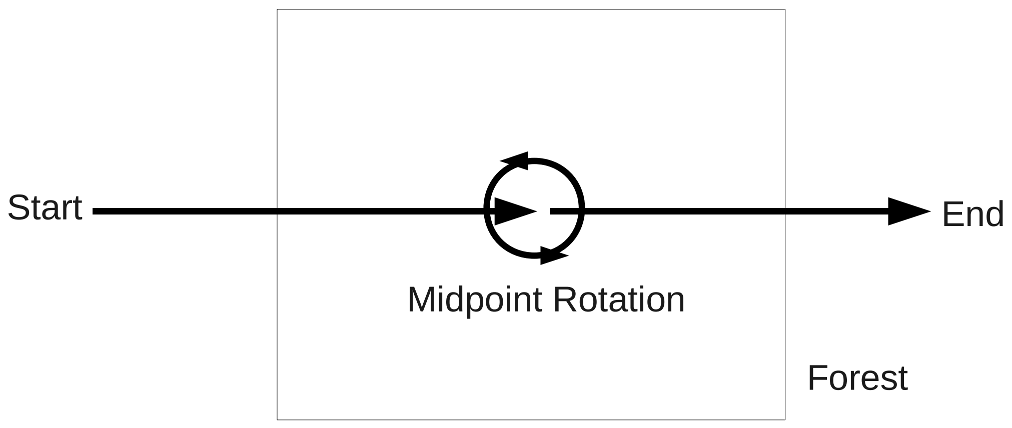


Illustration 11: The path that the camera followed through the forested area during the performance testing. This path is intended to provide a wide range of use cases for the frustum culling, ranging from no trees to cull (at the start), to all trees being eligible for culling (at the end).

In all cases, there was a directional force (wind) acting on all trees in the simulation. This was done to prevent PhysX's built-in optimizations from stopping the simulation of the trees, which occured when they fell into a rest state. The wind kept them moving and was also more indicative of the actual performance requirements for practical use of the system.

We only tested the optimizations that we ourselves implemented, and those included in the graphics and physics engines that were easily configurable. There were two main reasons for this. Firstly, because some optimizations included in OGRE and PhysX (the pre-existing graphics and physics engines we used) were heavily integrated into the engine cores and they were very difficult to disable or configure (such as frustum culling). Secondly, these heavily integrated optimizations are all well-established techniques that are known to work in practice. For this reason, testing whether they work would not be very useful.

In order to prevent any unpredictable, once-off factors from skewing our results, we performed multiple runs (twelve, to be precise) of each test condition with an identical set-up each time. The number of runs was originally meant to be 10 (which we felt was sufficiently large), but, due to a minor typing error, two extra runs were done for each case. Since these extra runs were perfectly valid, we chose to include them in our results.

In our final analysis, we used the average result, taken across all of the testing runs. Using the mean of a large number of runs averages out any unwanted factors that might manifest in any one of the testing runs.

The computer used for testing had the following specifications:

- Processor: Pentium 4 3GHz processor with hyper-threading
- RAM: 1GB
- Graphics Card: Nvidia GeForce 8600GT

During testing, all unnecessary programs and services were shut down so that they would not skew the results.

During the testing simulations, the time taken to perform each graphics update (drawing a frame) and each physics update (one step of the physics simulation) was recorded. These updates occurred sequentially in a single thread, but we felt that it would be beneficial to store the times separately. The times were recorded each frame, and at the end of the simulation, were written to disk. Both the CPU and wall times were recorded, but for analysis we chose to only use the CPU time, since it provides a better indication of performance than the wall time (which can be affected by other processes running on the system).

We felt that recording the time taken per complete simulation update would not be useful, as the physics and graphics update times overwhelmingly dominated the total time taken for each simulation update. Recording the times for graphics and physics separately was important in evaluating their relative significance and learning which (if either) was acting as a performance bottleneck.

4.3 User Testing

This testing was aimed at discovering how realistic the movements and dynamics of our trees were to the average user. The visual appearance of the trees was not considered at all. The best way to answer this question was to perform user testing with a sufficiently large and representative group of users.

Our testing was done in a cross-sectional study, using the final version of our system (after we had instituted a feature freeze). We recorded videos of our simulation running a number of situations that we felt represented a good range of our system's capabilities.

There were two reasons for showing the user videos instead of letting them interact directly with our system. Firstly, our system could not run large numbers of trees (more than 50) at a frame rate that was acceptable for real-time interaction. In spite of this, we wanted to test situations with large numbers of trees (even if not real-time, these are still relevant to off-line applications). Secondly, we wanted as little variation between user experiences as possible (it is important to control as many variables as possible across user tests). As an example of how a problem could occur, the navigation of a scene is dependent on the user's experience in navigating games or virtual environments. A more experienced user would thus have an easier time, and could rate our system higher than a user who was less experienced, affecting the results.

Pre-recording videos of the different simulations allowed us to solve both of these problems. The videos always ran at a real-time frame rate and were identical each time they were played.

The actual simulations that we recorded and used in testing were the following:

- An 'identity' case, where the only external force acting on the trees was gravity.
- Trees in directional winds of various strengths (ranging from a light breeze to hurricane force).
- Boxes of various weights being thrown at trees.
- Trees being felled and falling over.
- Powerful explosions occuring in a the middle of a group of trees.

- A strong tornado moving through a group of trees, uprooting some of them.
- A very strong attractive force placed in the centre of a group of trees, essentially acting as a gravitational black hole.

To prevent any bias from only using one type of tree in the simulations, we made videos of each situation with multiple tree types.

The number of trees in the scenes varied wildly, depending on the scenario in question, from one (in the case of the identity videos) to approximately two hundred (the explosion, tornado and black hole videos).

In order to give the users a fair basis for comparison we provided videos of similar events occuring in real life. This primed the user with a sense of how the events should look in reality, making them more likely to notice unrealistic behaviour and pick up on problems. In the case of the black hole and tornado, we were unable to source any appropriate real-life videos and hence had to ask the users to use their intuition to decide how realistic/appropriate these simulations were.

We also emphasized that the users should focus on the movement and dynamics of the trees and their interactions, not the visual appearance of the trees. This was due to concerns that users might otherwise blur the line between appearance and movement, and critique the visual appearance rather than the physics.

For each user we recorded relevant information about them, such as their age, experience with video games and field of study/work. This was done to allow looking for patterns correlated to any of these factors during analysis of the results. To see the exact details of what was recorded, please refer to the questionnaire attached in appendix B.

User evaluation of our system was done via a questionnaire. There was a section for each of the simulation situations described above, and a final section for the user to provide their overall impressions. All of these featured both qualitative and quantitative feedback. The quantitative parts were aimed at getting general impressions of the simulations, while the qualitative sections was intended to give the opportunity for specific comments and focused feedback.

The questions themselves were answered on a Likert scale of 1 to 7, with 1 being 'completely unrealistic', 7 being 'totally realistic' and 4 being equal amounts of realistic and unrealistic elements.

There were two overall quantitative questions in the final section of the questionnaire. These asked for the user's overall impression of the realism of our scheme, and whether they felt our trees were an improvement over those found in current video games. The scale on these questions ranged from 1 (current games are much more realistic than ours), to 5 (equal realism compared to current games), to 10 (a massive improvement over current video games).

Qualitative questions accompanied both the individual scenario questions and the overall questions. These provided an opportunity for the user to expand upon, and qualify the ratings they gave.

The questionnaire and all supporting documentation is included in this report in appendix B.

The different situations (the scenarios listed above) were shown to each user in a random order. This was to average out any bias that might arise from a specific order of the videos.

All testing was done in a controlled environment, following all the necessary requirements for user testing, by controlling all independent external variables during the testing. This meant that everything was kept as identical as we could manage for the different users; the person administering the test interacted with the test subject as little as possible, the area where the testing occurred was kept the same for different tests and pre-created scripts were used when instructing the user in how to do the testing.

The testing was done in tandem with similar testing being conducted for research into adding physics components to procedurally created buildings. This meant that the users were shared between the two tests. In order to counter any learning effects that might occur as a result of this, users alternated between doing our tree testing first, and doing the buildings testing first. In this way, any bias that might occur from being exposed to one before the other would be averaged out in the results.

The main testing was done with 20 users. There were, however, two additional tests that occurred. One of these was a pilot test, used to ensure that all of our procedures were sound. Since we made some minor changes to our procedure, based on this pilot test, we could not use the results.

The second additional test had a power blackout occur halfway through it. Since the test was incomplete, and resuming it later would break the controlled environment we were using in our tests, we discarded the results from this test.

Finally, we considered ethical issues around this testing and took these into account. For a description please see appendix A (specifically section 9.1.5).

4.3.1 Testing Procedure

The testing was all performed in the computer science honours lab at the University of Cape Town. The procedure was as follows:

1. The user was given the introduction sheet to read (this explained the process and gave them the necessary information).
2. The user filled out an information sheet with information about themselves (age, video game experience, etc.).
3. The user undertook an evaluation of the first situation (the order of the simulations was randomised for each user)
    a) The user watched real-life videos. They could watch these as many times as they wished.
    b) The user watched the videos of the simulation. They could watch these as many times as they wished, but they could not go back and look at the real videos.
    c) The user filled out the quantitative and qualitative evaluations for that particular situation.
4. The user repeated step 3 on the rest of the simulation situations, in the randomised

order generated for that user.

5. After all the simulation situations had been evaluated by the user, they filled out an evaluation of their overall impressions and feedback on the system.
6. The user was given the option of using the actual simulation software itself. If they took up the offer, the test administrator made a note of any feedback the user gave about their experience with the actual program.
7. The user was thanked and paid for their participation.

4.4 Visual Heuristic Validation

In this phase of evaluation we aimed to analyse our generated trees, in comparison to real-life trees, and examine the level of realistic and unrealistic behaviour they exhibited.

The main differences between this and the user tests were:

- This evaluation was conducted by the author instead of by users.
- We were comparing against real-life trees instead of videos of real-life trees, which meant that a much wider range of subtle differences could be examined.
- Here we were interested in how realistic our trees actually were, instead of how realistic they appeared to be.
- There was no numerical rating, all the feedback was qualitative observations.

This meant that this was a much more thorough and rigorous version of the user testing, aimed at discovering notably unrealistic aspects of the trees.

Ideally, we would have preferred to have done a full analysis based on the laws of physics (momentum, acceleration, mass, etc.), but that was not an option due to time pressures and our lack of the required knowledge to perform such a detailed analysis.

This testing was heuristic in nature - we did not evaluate based on any scientific criteria (such as equations of motion or momentum calculations). Rather it was based on searching for elements that stood out as being especially noticeable (either realistic or unrealistic).

Since we were conducting the testing, and had intimate knowledge of some of the problem areas of the system, we could also use this opportunity to take a step back and evaluate how badly those areas impacted on the final result.

4.5 Conclusion

We have presented the design and rationale of the experimentation performed in our research. The two main focuses were evaluating our system's realism, and computational performance through two types of testing performed under controlled conditions. We also explained the third component of our testing: a heuristic based analysis performed by the author.

5 Results and Discussion

5.1 Introduction

After the experimentation had been completed, and the raw data recorded, we were able to analyse the results to see how well our system performed in the various experiments.

It was not necessary to remove any outliers because the testing data was all within expected bounds and there were no suspiciously large or small values that would indicate a need for cleaning.

The structure of this chapter follows that of the previous one, covering performance testing, user testing, visual heuristic validation and finally, our overall conclusions from the results.

5.2 Performance Testing

The first step in analysing our results was to average the various runs together and plot the results graphically to determine the growth rate and orders of magnitude involved. Note that the raw data from all of our performance tests can be found in appendix C.

The number of polygons per tree varied due to the random nature of stochastic L-systems. On average, though, it was roughly 250, never dropping below 100, and seldom rising higher than 750.

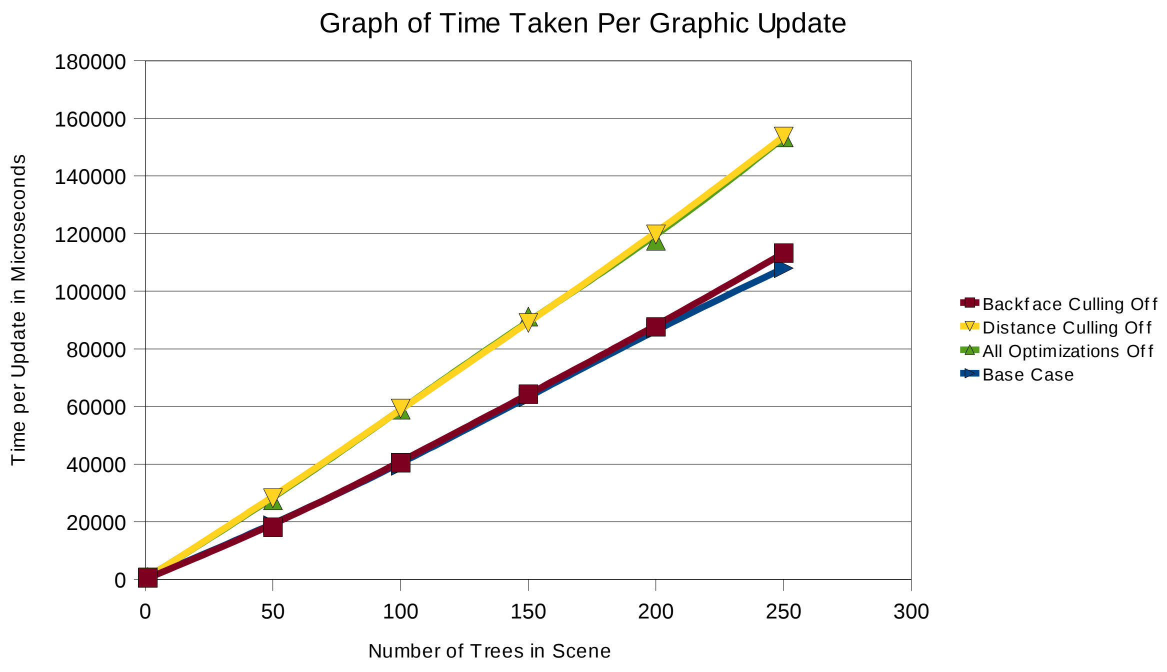


Illustration 12: Graph of the average times to do a graphics update, taken across all testing runs. Note that the times fall into two bands, with distance culling being the differentiating factor between them. Aside from that, there is little variation between the times.

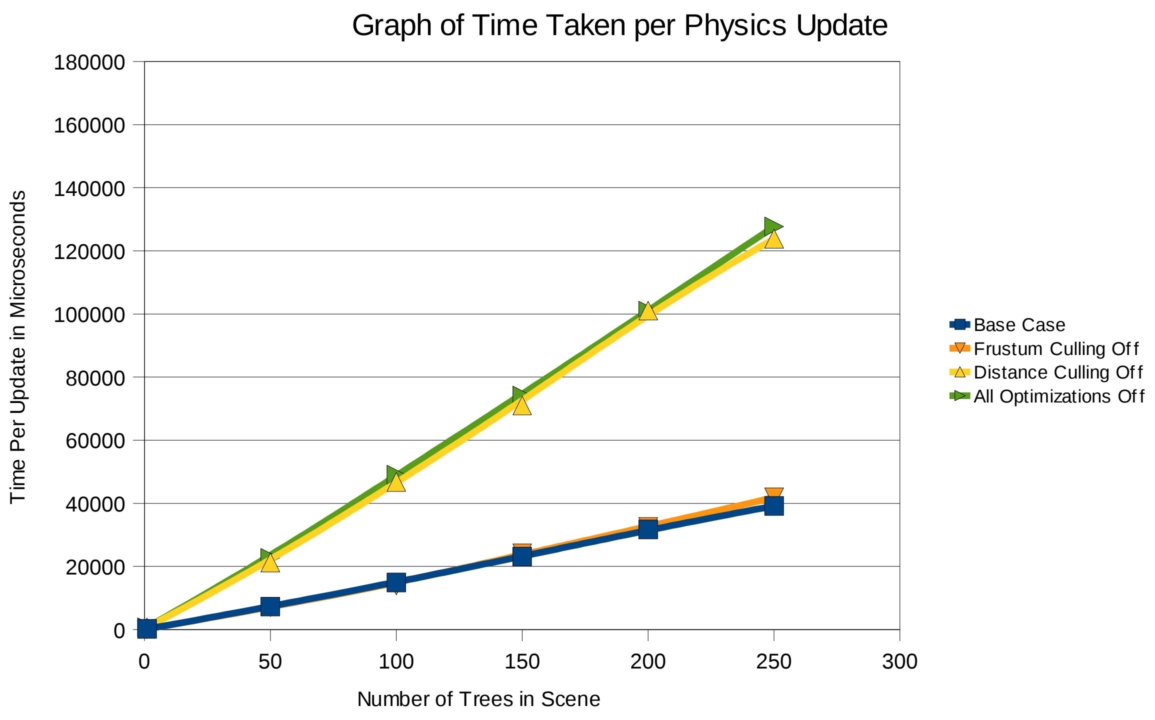


Illustration 13: Graph of the average time needed to update the physics simulation, taken across all testing runs. Once again, we see the two bands emerging, with distance culling being the differentiating factor.

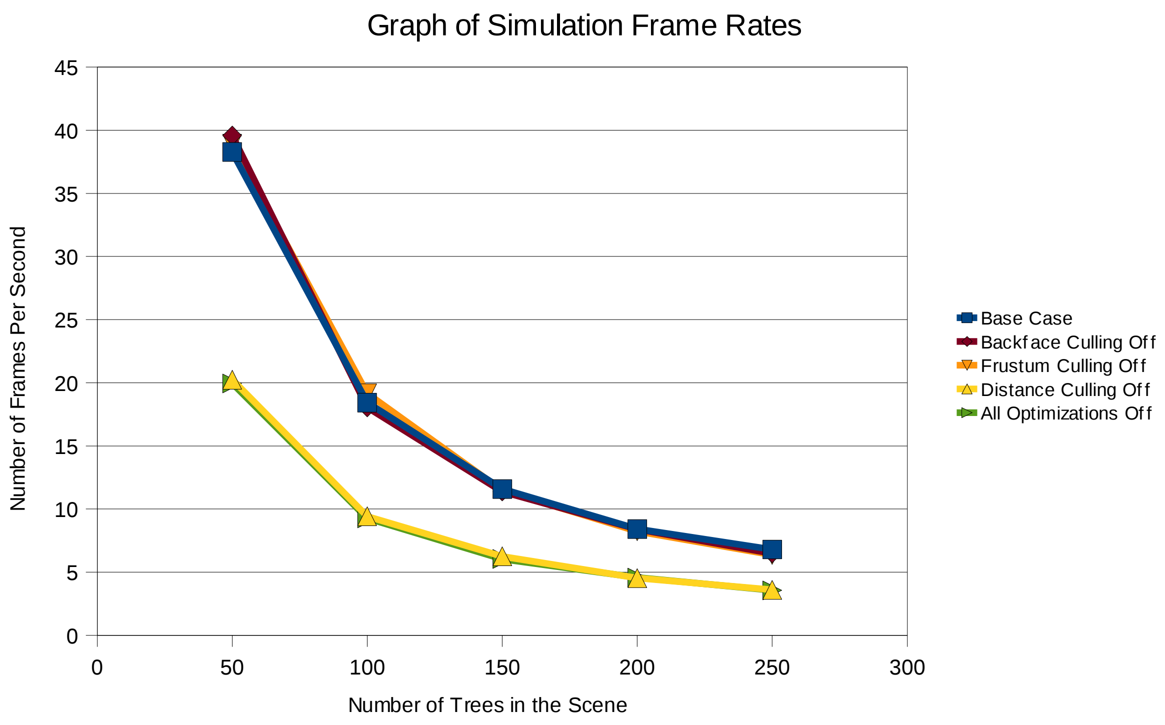


Illustration 14: Graph of the actual frame rates from the simulations. These were obtained by combining the graphics and physics times. Note that the values for 0 trees have been left out because they were all extremely large and made the graph difficult to read. The most important result in this graph is that a real-time frame rate is only produced for 50 trees (or less) with distance culling enabled.

Note also, that the rate of growth for both the graphics and physics times is linear. This is a pleasing result, but expected, since a larger rate of growth (say quadratic or cubic) would make our algorithm impractical for real-time use.

We can also observe that in the base case (i.e. all optimizations turned on), the time to render the frame is much larger than the time taken to update the physics, by a factor between 2.5 and 3. This strongly suggests that, as our system stands, the bottleneck lies on the graphics side of things, and that in trying to improve performance, this should be the first place to look.

By looking at the graph of the frame rates, we can clearly see that a real-time frame rate was only possible with 50 trees (or less) and distance culling enabled. This graph also emphasizes that distance culling is the most effective of the culling schemes, as it is what separates the results into the two distinct bands of performance displayed in the graph.

5.2.1 Distance Culling

Looking at the optimizations themselves, we can see that the most effective is distance culling. When distance culling is disabled, it causes the largest increase in time per tick for both graphics and physics. This makes sense as it has the largest potential for cutting out geometry or physics primitives. As can be noted from the graphs, it speeds up the graphics by a factor of roughly 1.333, and the physics by a factor of more than 2. For this reason, distance culling seems to be the most important culling scheme, providing the largest performance bonus. This is in line with distance-based billboarding techniques.

However the following points should be borne in mind:

- As an approximation of billboarding, distance culling fails to account for the cost of drawing the billboards. This cost may lessen the gain provided by distance culling.
- The effect of not simulating distant objects (as distance culling does) on realism and immersion has not been tested sufficiently for us to conclude that it is appropriate. This effect depends on the distance threshold used for culling (see section 3.6), and it may be that it is not an appropriate optimization in practice, due to a lack of realism.

5.2.2 Backface Culling

Looking at the graph we see that disabling backface culling does not seem to make much difference to the frame rate of the program.

In fact, the difference between having it turned on and off is virtually unnoticeable for smaller numbers of trees, and a noticeable decline in frame render time only occurs for more than 200 trees.

After considering this we came to the conclusion that this was to be expected due to the relatively small amount of geometry that would be eligible for backface culling. The problem is that the leaves in our trees are two-dimensional rectangles which cannot be backface culled (or the leaves would be invisible from one direction). Hence backface culling was disabled for the leaves, leaving only the cylinders that form the tree trunk and branches. Since the leaves outnumber the branches (by a factor of about 2), and are by far the most visible part of the tree, it can be concluded that there is relatively little geometry that is eligible for backface culling, hence the small performance loss exhibited by removing it.

5.2.3 Physics Frustum Culling

The physics frustum culling, like backface culling, seems to make little difference to performance based on these results. This was initially puzzling, since it seems that the cost of doing a frustum collision check should be significantly less than the cost of a full physics tick for a tree.

We followed up on this by carefully checking our code for any errors, and looking into what might cause such odd behaviour. Our reviews found no errors in the code; ultimately the answer lay in the internal workings of the PhysX engine: PhysX identifies objects that have not moved (due to falling into a rest state) and 'puts them to sleep' (ceasing to simulate that object until some external force 'wakes it up'). In our initial testing run, there were no external forces acting on the trees, which meant that the trees were automatically put to sleep. Hence, most of the time, any objects that could be excluded from simulation by being outside the viewing frustum were already asleep, and there was no performance gain from re-signalling sleep mode.

The obvious solution to this was to create a force that acted on all of the trees during the performance tests. We implemented this and re-ran the tests, but the results were similar. With some further investigation we discovered that continuous forces acting on the trees woke any sleeping objects up, so as soon as they were switched off by the frustum culling, they were immediately reactivated.

Unfortunately, there was no easy fix to this problem. We looked for a method to totally disable the simulation of an object, but were unable to find one. Solving this problem would have required delving into the source code of PhysX itself, which is not freely available for non-commercial uses such as our research. This hurdle, combined with time constraints, meant that we were forced to leave our investigation of physics frustum culling at this point.

Nonetheless, even in a crippled state, frustum culling does seem to provide some marginal speed improvements in our implementation (noticeable in the graphs for more than 150 trees). We firmly believe that if properly implemented (totally disabling the simulation of invisible objects) it could provide significant performance gains. For this reason we recommend that any future work examine this feature and it's potential for increasing performance.

5.2.4 Performance Testing Conclusions

There are two main conclusions to draw from the results of our performance testing. Firstly, our system is not capable of real-time performance for more than 50 trees. A frame rate of 30fps requires that each frame is rendered in 33 000 microseconds or less. As can be seen in our graphs, for more than 50 trees (roughly 1500 physics capsules and 30000 polygons), the combined graphics and physics times exceed 33 000 microseconds, hence falling outside a real-time frame rate.

The second conclusion to draw is that optimizations and culling schemes are important, and can increase performance dramatically, as is most visibly demonstrated by performance gains from distance culling.

5.3 User Testing

5.3.1 User Information

We began analysis of our user testing by looking at the information about our various test subjects. A complete breakdown of the user information can be found in appendix C, but the important points are presented here.

Most of our test subjects were in their early twenties, with the average age being 21.7, the youngest being 19 and the oldest being 24. Our gender breakdown was roughly two thirds male and one third female (14 male, 6 female).

The most common field of study was overwhelmingly computer science (17 out of 20), but some other fields were represented (psychology, philosophy and actuarial science).

There was a wide range of gaming experience among our test subjects, ranging from virtually no experience, up to regular gaming during the week and honours level study in video game programming. This provided an opportunity to examine whether gaming experience was a factor in the perceived realism of our trees. One striking point, however, was that all users had been exposed to games for at least four years.

5.3.2 Quantitative Feedback

Having obtained a feel for who our users were, we moved onto looking at the quantitative components of the user testing. Once again, the raw results are included in appendix C.

For the purposes of this scale, we considered 5 to be the lowest rating at which the trees would still be acceptable for actual use. At this level the trees were realistic but contained a noticeable amount of unrealistic elements. This meant that if the average rating for a particular simulated situation was 5 or more, the aspects of tree behaviour in that simulation were acceptably realistic.

As can be seen from the above graph, 6 out of 7 scenarios met the requirement of a mean of 5 or more, in comparison to reality and the realism expected from a video game. The one scenario that did not was the explosion situated in the middle of a group of trees, where the rating against reality was only 4.2.

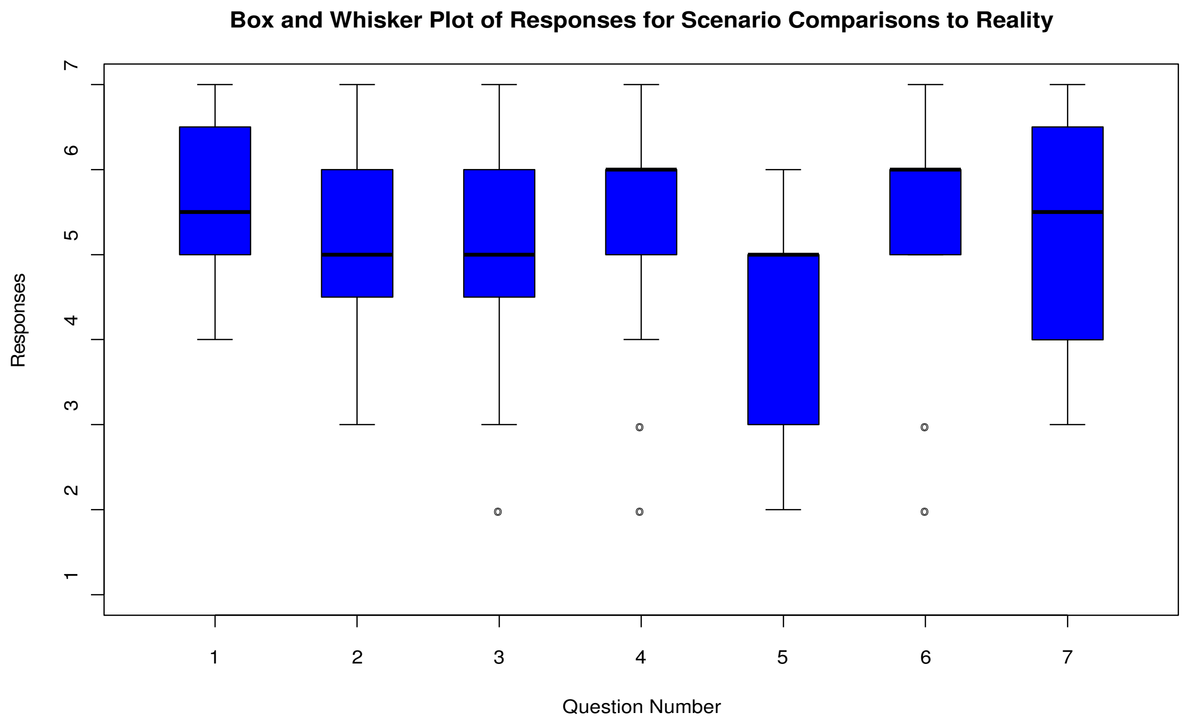


Illustration 15: Box and whisker plot of the user responses when asked to rate the testing scenarios against reality. All median values were 5 or more and, only in the case of scenario 5, do any of the lower quartiles drop below 4. This suggests that, our system held up well when compared to reality.

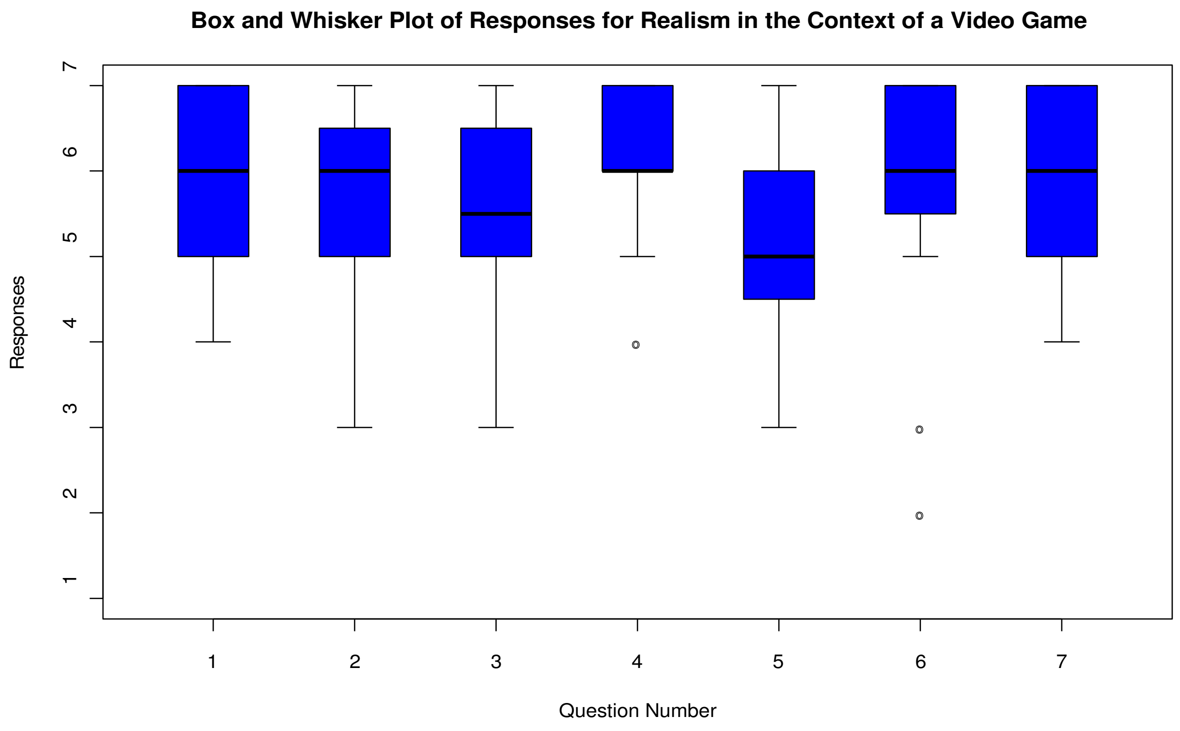


Illustration 16: The box and whisker plot of user responses when asked how appropriate they felt our system was for use in a video game environment. The results are visibly higher than those comparing our system to reality, which was expected, and suggest that users believe our system would be very realistic when put in the context of a video game.

The standard deviation of the ratings across all of the individual questions was 0.46. This suggests that the user ratings are reasonably consistent, since they are focused around an area on the range of answers. If the standard deviation had been higher, it might indicate the ratings were very spread out and inconsistent, which would make our results inconclusive. However, the standard deviation is within acceptable boundaries, and hence we do not have this problem.

It can also be noted that in all cases, the average score for the comparison to reality was lower than the score for a gaming context. This is a result that we were expecting, considering that our tree definitely do not look as visually realistic as real trees. If it had been the other way around (scores compared to reality being higher than appropriateness in games) then that would have been cause for concern.

Next, we moved on to examine the overall quantitative scores. This section was aimed at the users' overall opinion of the realism of the trees' physics, rather than for individual components.

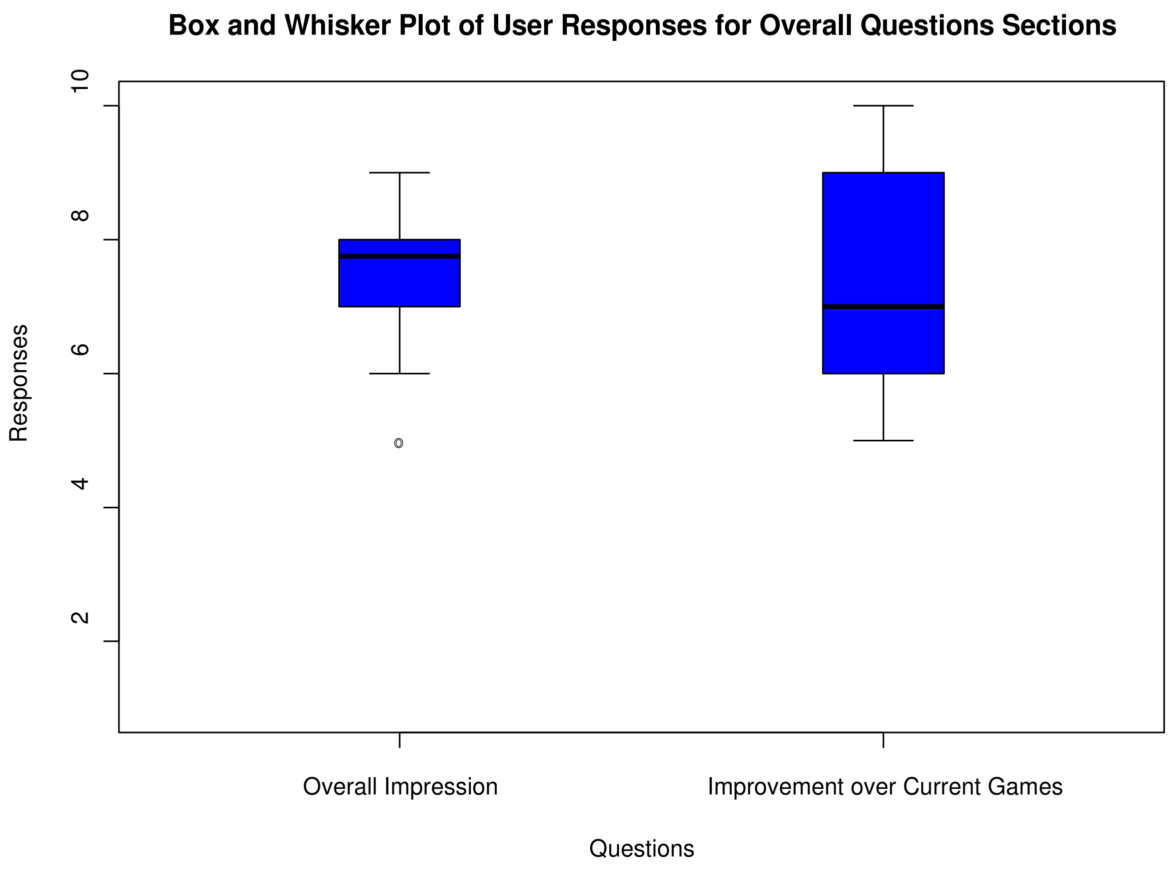


Illustration 17: Box and whisker plot of user responses to the final section of the questionnaire, where users gave their overall impressions of our system. As can be seen, the responses were positive, with none of the users rating our system as being worse than the trees in current games.

The overall mean score score of 7.38 indicates that users found our trees to be roughly halfway between being 'realistic but with noticeably unrealistic elements' and 'totally realistic'. This is an encouraging score as it indicates that the overall impression our trees left in the minds of users was a convincing one. The standard deviation for this score (1.15) is relatively small, suggesting that most users agreed on this value, which correlates with the small interquartile range for this question, as shown in illustration 17.

It is also worth noting that this average score correlates very closely to the average scores from the individual sections (relating to the realism of the different aspects of the trees' behaviour). The overall score's average is 7.38 and the average of the individual scores is 7.33 (when converted to a scale of 1 to 10), implying strong internal consistency.

Not all of the test subjects answered the second quantitative overall question (whether they believed our system was an improvement over the trees found in current games). Three of them opted to leave the question unanswered as they felt they did not have the experience necessary to provide an answer. The mean score taken across the 17 users who did answer this question was 7.24.

In many ways this was the most important result of our research, since it suggests that our trees represent a significant improvement over those found in current video games. It also vindicates our assertion that adding physics components to trees would make them more realistic and immersive.

The standard deviation of this question was 1.8, which is noticeably higher than the other questions. However, it still implies the different answers were closely distributed enough that the majority of users agree our system is an improvement over those in current video games. This is further supported by the interquartile range shown in illustration 17.

This overall score is also slightly lower than the average scores across all the individual scenarios questions (7.24 overall versus an average of 8.12, when converted to a score out of 10, across the individual sections). Using a t-test[17], we determined that the overall score is indeed lower than the average of the individual question scores (with a t-value of 3.197).

This difference suggests that the users considered the individual components more realistic than their overall impression of the system. We suspect this was due to one or two of the scenarios that the users found less realistic (possibly the explosion scenario, as it seemed to score the lowest) leaving them with a bad impression of the overall realism. This bad impression would cause the user to rate the overall realism lower, despite finding the individual components very realistic.

5.3.3 Qualitative Feedback

Following our analysis of the quantitative data, we examined the qualitative feedback that accompanied it. Most subjects wrote extensively, especially in describing the areas that they felt were unrealistic.

There was far too much written commentary to include in this report, but after reading through all of the feedback we noted which ideas and criticism occurred the most frequently. Here we briefly highlight those points.

17 http://www.socialresearchmethods.net/kb/stat_t.php

Please note that we were aware of some of these issues before the testing. Those issues are presented and discussed in more depth in section 5.4 (Visual Heuristic Evaluation). We have also included some frames taken from the videos to provide an idea of their format (illustrations, 18, 19 and 20).

There was a pleasing amount of positive comments. One user commented that the movements and dynamics of our trees were comparable to Crysis[18] (a relatively recent first person shooter video games that roundly received glowing praise for its visuals and realism), and another described the realism as 'astounding'. These were not focused on particular areas, but do show that at least some of the test subjects were impressed by the results.

The problem that most users picked up on was that some of the branches of trees began doing violent windmill motions which did not stop, spinning around at high speeds in a completely unrealistic manner, that was incongruous with the movements of the rest of the tree.

This was a known problem, which is discussed in more detail in the visual heuristic section (5.4) below. The problem randomly occurred in trees whose generative L-systems met certain criteria, and stemmed from a number of sources, with the end result that lighter branches in the tree could end up in self-propagating perpetual motion. We were unable to find a satisfactory solution for the problem during development and, hence, we expected this to be noticed by users.

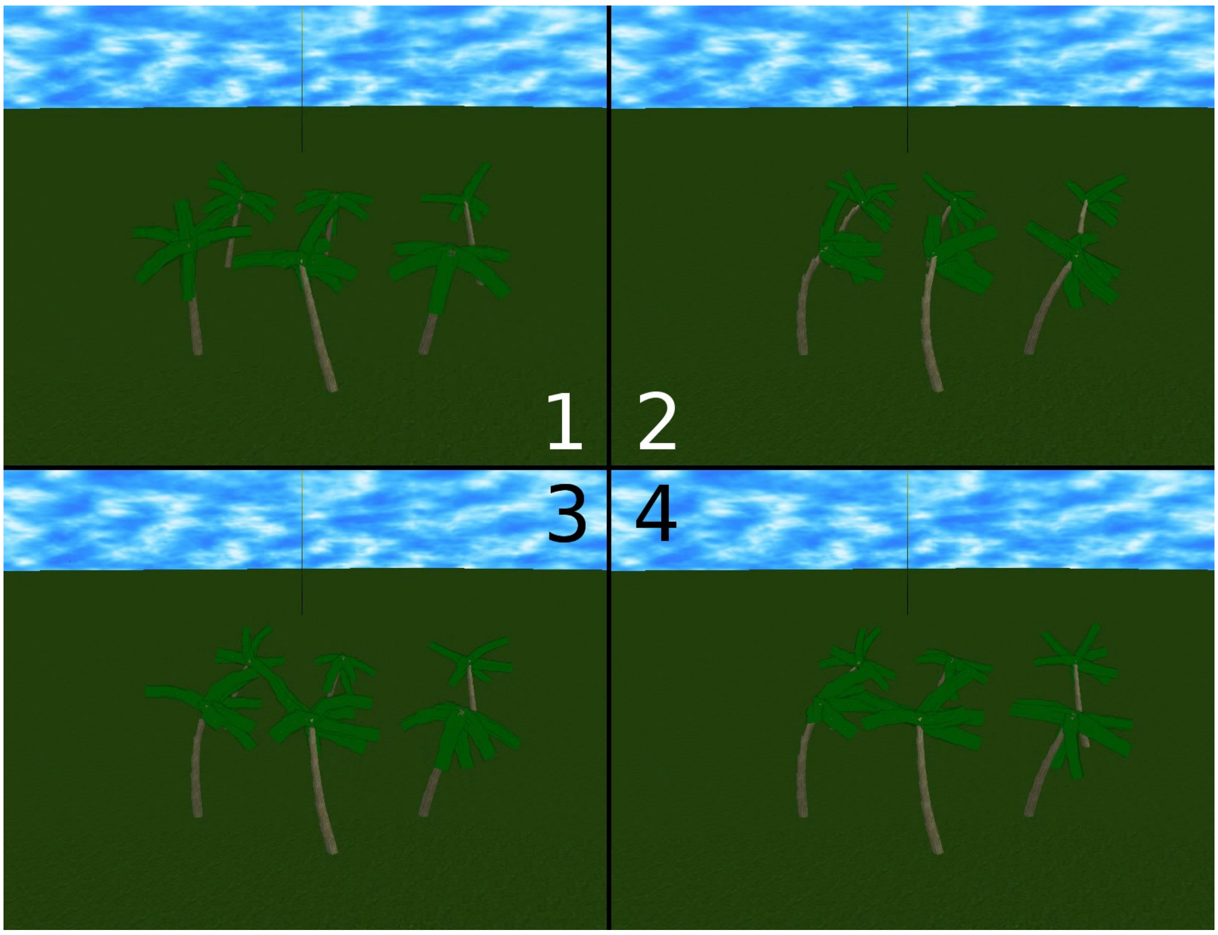


Illustration 18: These are an ordered set of frames (though they are not sequential) showing a group of palm trees experiencing a moderate wind.

18 http://www.ea.com/crysis/

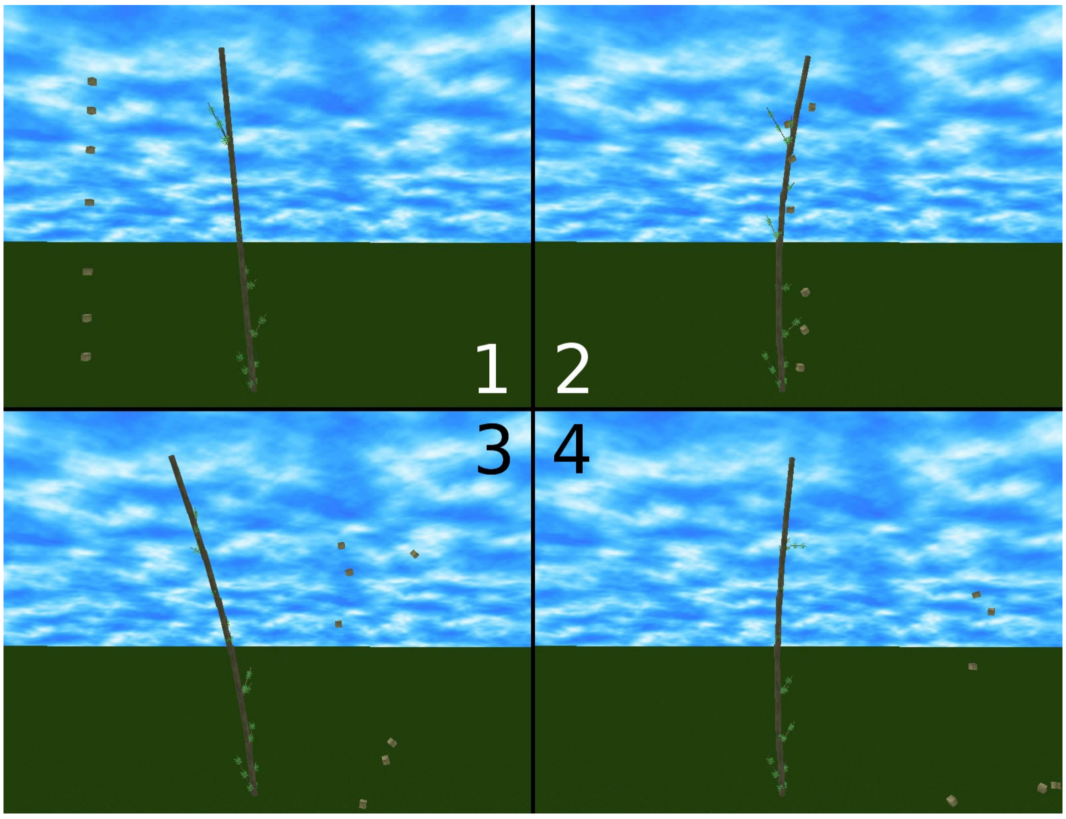


Illustration 19: A montage of frames from one of the user testing videos, which consisted of a number of boxes being thrown at a bamboo tree, and the ensuing movement it displayed.

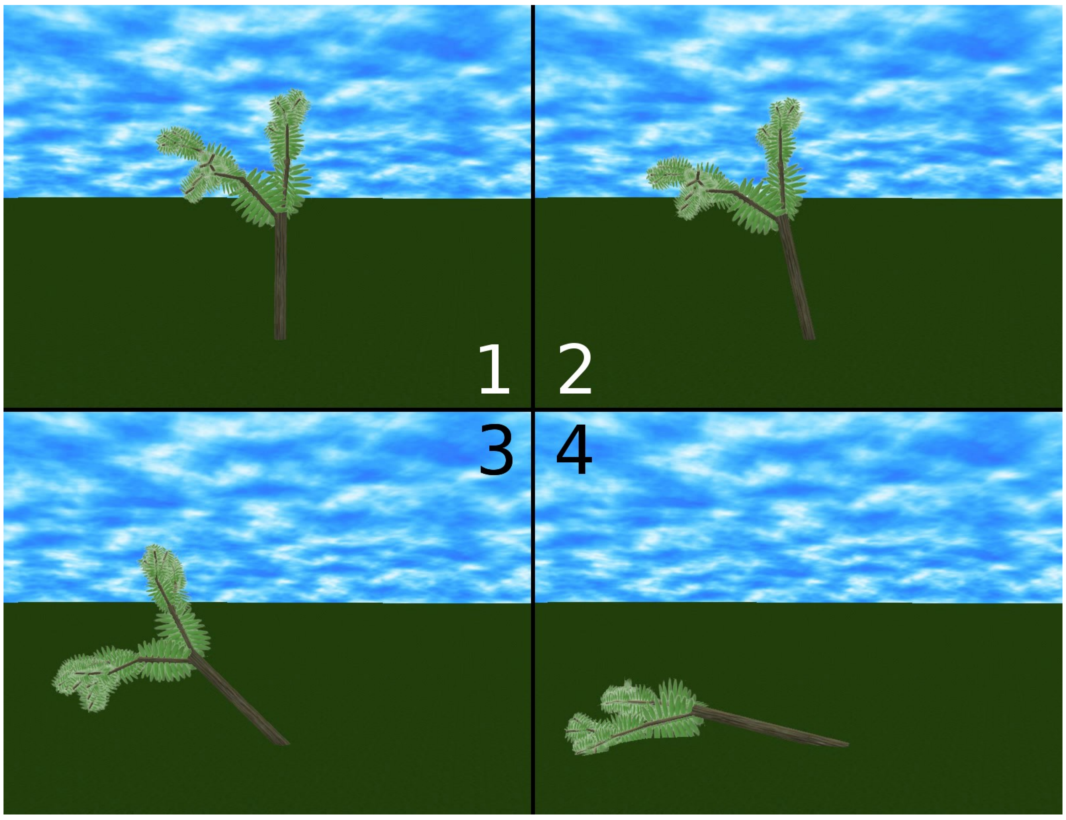


Illustration 20: Four frames from a video that displayed a tree falling over.

The next most common criticism among users was that the leaves of the simulated trees did not exhibit the fine grain behaviour shown in real trees, such as each leaf moving independently in slightly different directions.

Once again, this was a known limitation of our system, as discussed above in section 3.7. We were aware of this issue and expected feedback on it.

Another commonly raised set of points was that the forces involved in our simulations were subtly unrealistic. This covered several complaints such as the wind we used being too uniform (real wind is bursty), our tornado approximation not having a strong enough force in the centre and too quick a drop off, and the gravity being noticeably lower than it is in reality. These comments were unexpected. On the one hand, they are valid criticisms that negatively affected the realism of our system, but on the other hand, they are configurable, and not an issue with the core idea and algorithm of our system.

Some of the simulation situations also drew the criticism that the trees in the simulations were not sufficiently rigid. Feedback on this was most prevalent in the scenario of boxes being thrown at the trees, but also appeared in some of the other scenarios. The complaint was that the simulated trees were far too springy, and moved much more than trees in reality did when hit by objects.

Linked to the above point, many of the users felt that the motion of our trees were not dampened enough. This meant that when they were moving, they took too long to slow down to a halt. It should be noted though, that this problem is dependent on the L-system used to create the trees, and not our algorithm itself.

There were many other criticisms and observations from the test subjects, however, the ones above were the most pervasive ones, appearing a noticeable number of times, making them more likely to be valid points. We chose not to include the less common criticisms as they were far more likely to be biased by personal preference.

5.3.4 User Testing Conclusions

Based on quantitative feedback we conclude that our trees would be acceptably realistic for use in video games, and, in fact, were seen as an improvement over the dynamics of trees in current games.

The qualitative feedback reinforces this idea, but highlights many areas for improvement, which should be addressed before this system is used in practice. None of these problems are critical though, and many are dependent on the particular L-system used to produce the trees, not on the the fundamental ideas shown in our system. Unfortunately, it is not easily possible to modify the system so that no L-systems would exhibit this behaviour without losing a large amount of the flexibility and generality of the L-systems.

5.4 Visual Heuristic Validation

During this validation we identified several areas of unrealistic behaviour in the generated trees. Some of these were issues that the users picked up on. In the case of those, we present a more in-depth explanation of them here than was provided in the user testing section.

5.4.1 Branches exhibiting Perpetual Motion

The biggest and most obvious problem that plagued some of our trees was smaller branches getting stuck in perpetual motion cycles. This meant that they swung around at high velocities in an obviously unrealistic manner, often impacting nearby branches that were not exhibiting the behaviour.

We noticed this problem early in development, and began investigating its cause. There seem to be two separate reasons why it may occur. Firstly, it can occur when multiple physics 'bone' segments overlap. This can occur due to the random nature of L-systems when placing branch segments. The two bones will be forced apart as soon as the simulation begins, but the combined forces of trying to occupy each other's space and the springiness of the joints holding the segments together combine to form unending violent motion in the two branches. See illustration 23 for an explanatory diagram of the problem, and illustration 21 for an example of the effect this can create.

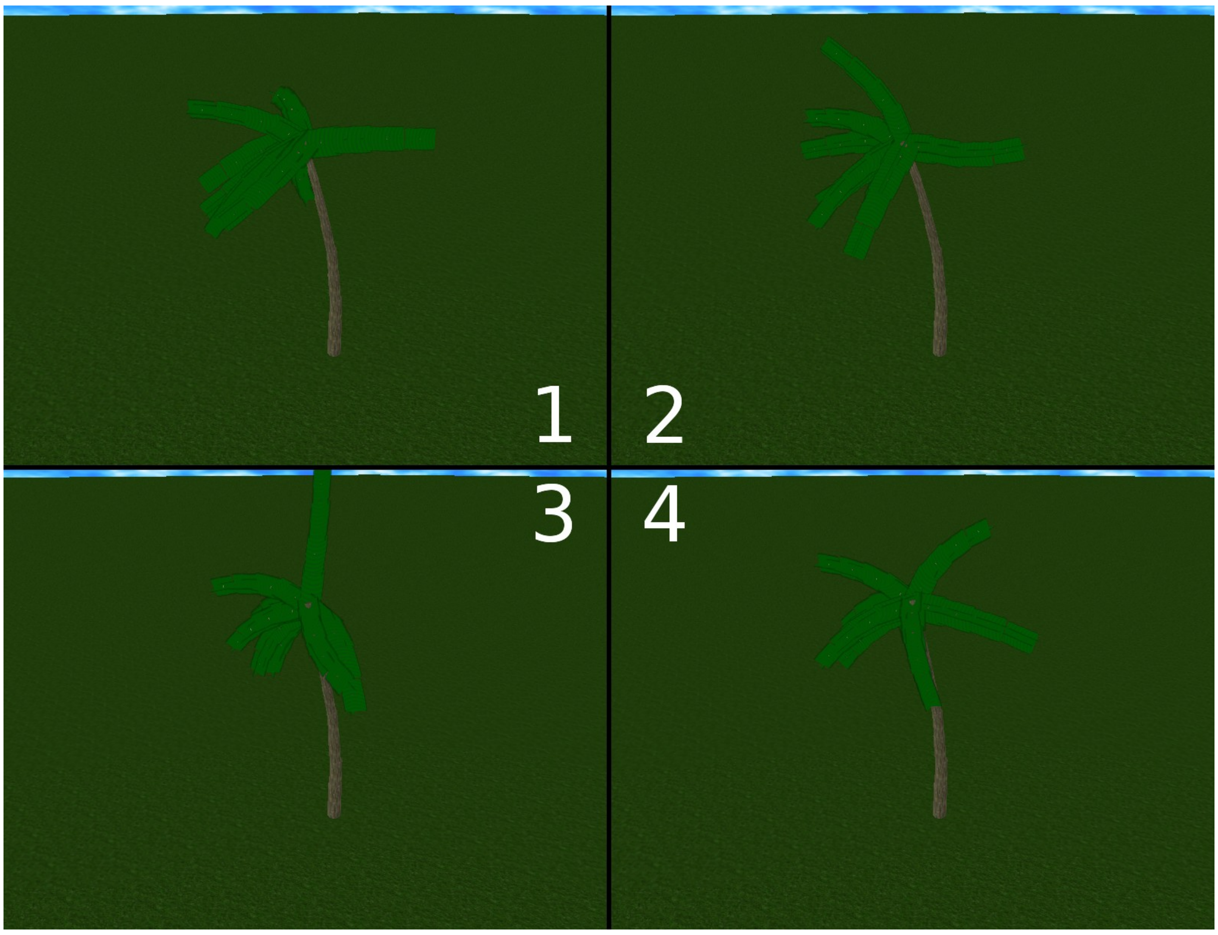


Illustration 21: A series of frames showing the perpetual motion problem occuring in a palm tree, due to the branches of the palm tree being created in overlapping positions. There were no other forces acting on the tree except gravity. These frames were recorded in quick succession, which indicates the high movement speed of the branches.

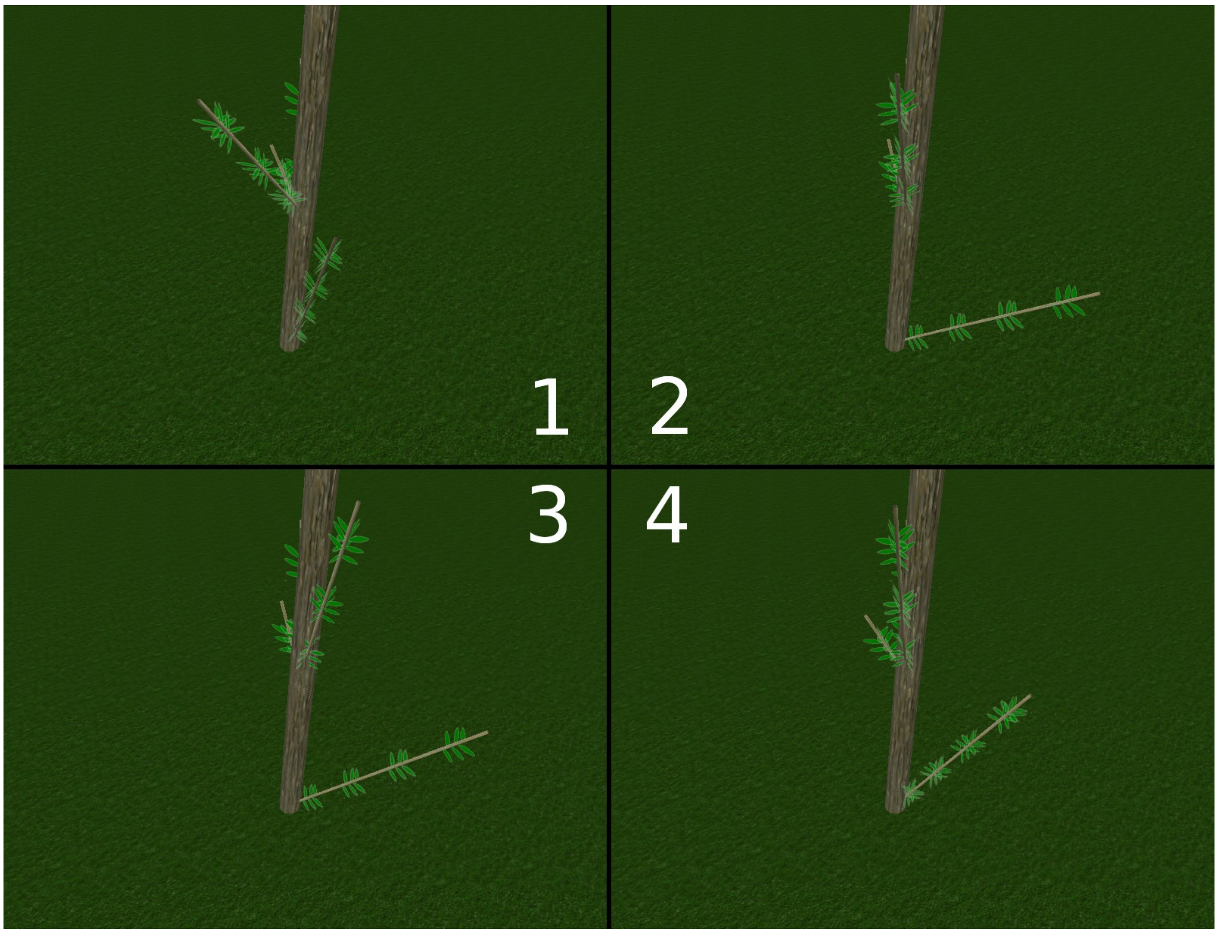


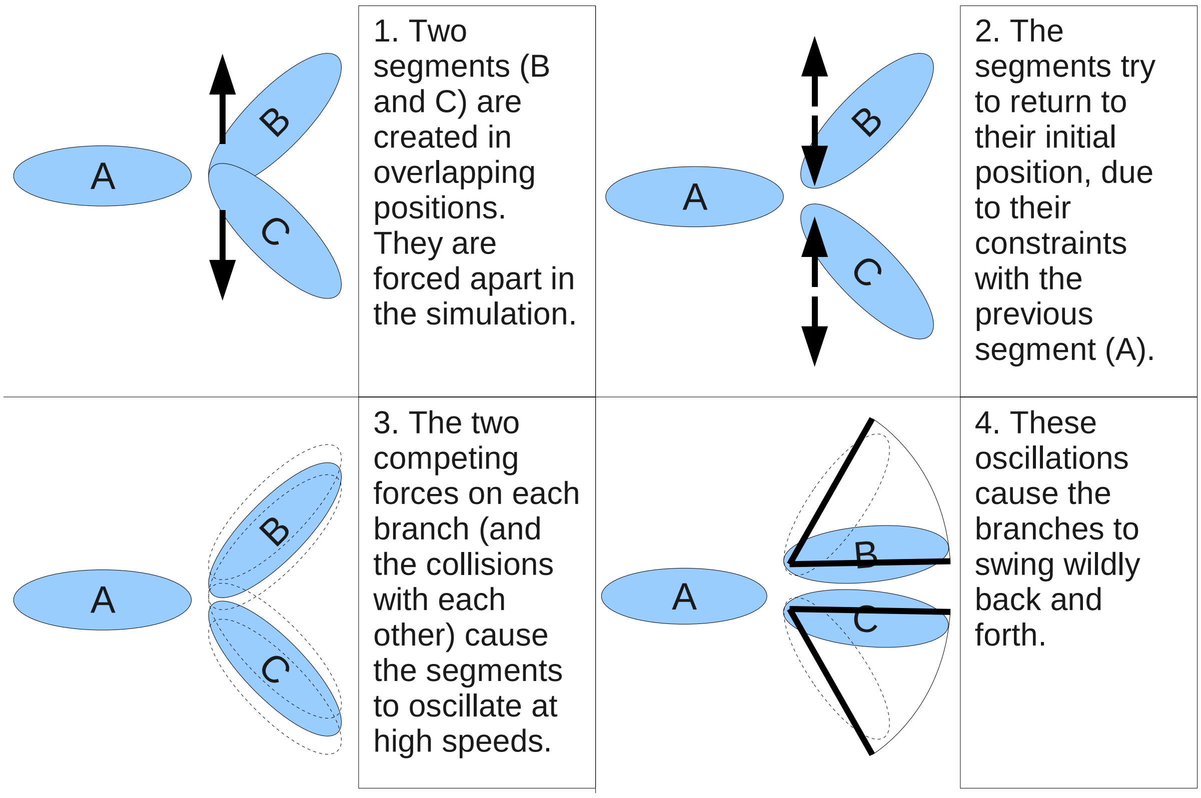
Illustration 22: Four images showing the effects of a narrow cone of movement on a light branch attached to a heavier one. In this case, bamboo stalks growing out of the main trunk. The stalks in these frames are all rotating in circles at unrealistically high speeds.

This problem can be solved by disabling the spring constraints in the joints, but this is not suitable for all types of trees.

The second way that this can occur is when a light branch is connected directly to a much heavier branch with the joint spring enabled, and the cone of allowed movement of the lighter branch is very small. Refer to illustration 22 to see the problem in practice.

As the lighter branch reaches the edge of it's cone of movement it gets moved back with a slightly springy force. This spring force causes it to collide with the other side of its movement cone, making it spring back again. This is repeated over and over, gradually building up a momentum that is constantly replenished from the collisions. See illustration 24 for a diagram showing this process.

This behaviour also occurs when the two branches are of an equal weight, but since the collision spring force is small, it is not sufficient to produce much movement in heavier branches and the dampening force has a chance to negate it.



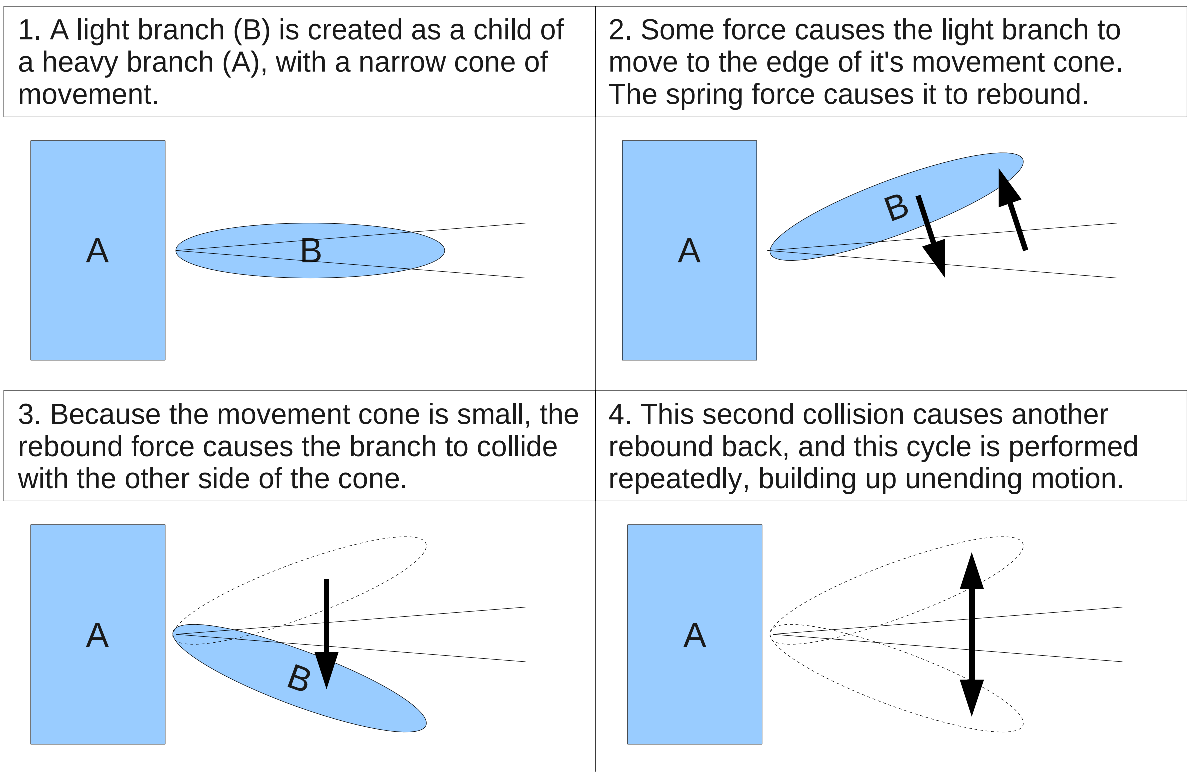
1. Two segments (B and C) are created in overlapping positions. They are forced apart in the simulation.

2. The segments try to return to their initial position, due to their constraints with the previous segment (A).

3. The two competing forces on each branch (and the collisions with each other) cause the segments to oscillate at high speeds.

4. These oscillations cause the branches to swing wildly back and forth.

Illustration 23: A series of images showing how the overlapping branches can cause the perpetual motion described. Illustration 21 shows this problem in practice.



1. A light branch (B) is created as a child of a heavy branch (A), with a narrow cone of movement.

2. Some force causes the light branch to move to the edge of it's movement cone. The spring force causes it to rebound.

3. Because the movement cone is small, the rebound force causes the branch to collide with the other side of the cone.

4. This second collision causes another rebound back, and this cycle is performed repeatedly, building up unending motion.

Illustration 24: A series of images showing how a small movement cone can cause a perpetual swinging motion in branches. Illustration 22 displays a manifestation of the problem in our system.

5.4.2 Behaviour Under High Forces

The second most noticeable instance of unrealistic behaviour was the separation of branch segments under high forces.

This problem occurred due to the discrete nature of the simulation. The simulation is composed of many short time steps that approximate continuous motion. In the case of extremely high forces, the amount moved during a time step may be much larger than expected. This means that an object might get pushed outside a constraint on it in one time step, rendered that way for a frame, and only be moved back inside the constraint on the next time step. This creates a scenario where objects could be displayed in positions that they are not meant to be in.

This issue is related to the problem of collision detection for fast moving objects, which is a widely known problem in discrete simulations [3] In our system, the problem occurred with branch segments being forced apart (outside the distance constraint between them) due to very high forces in the simulation.

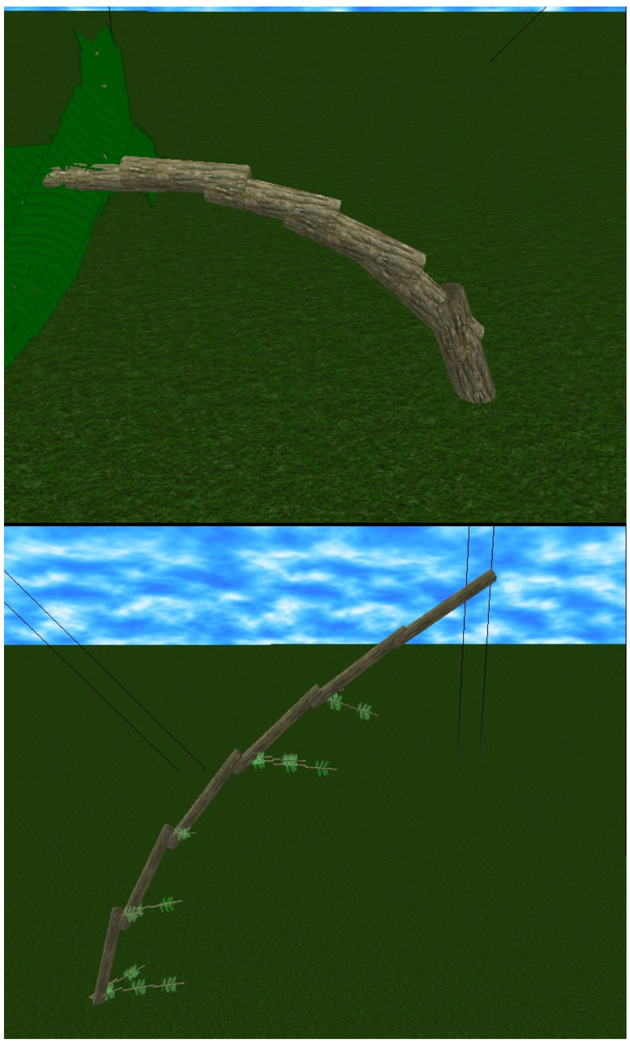


Illustration 25: Two examples of how high forces can distort the trees to separate the segment cylinders, which causes visually unrealistic results. The constraints on the segments are unable to cope with the magnitude of the forces applied on the tree, meaning they move far enough to be visibly apart.

This problem lies with the physics engine itself, and not with our algorithm. There is not much that can be done about this problem, short of enhancing the physics engine's collision detection and constraint handling. Sadly, such a project was beyond the scope of this research.

Luckily, such high forces are unlikely to be used in practice, and were only used by us in stress testing of our system. Additionally, if they were used in practice, it is likely that the trees would be made breakable, so that instead of trying to hold the branch together in an unrealistic manner, the joints would break, causing the tree to fall apart. For these reasons, we are not overly concerned with the problem of very high forces in the simulation.

5.4.3 Long Branch Segments

The other major issue that we observed was that long branch segments, as occurred in some of our generated trees can appear unrealistic. The issue here is that each branch segment is displayed as a cylinder. In reality, sections of tree branches are not uniformly smooth cylinders. They vary in shape and width, and can curve along their length.

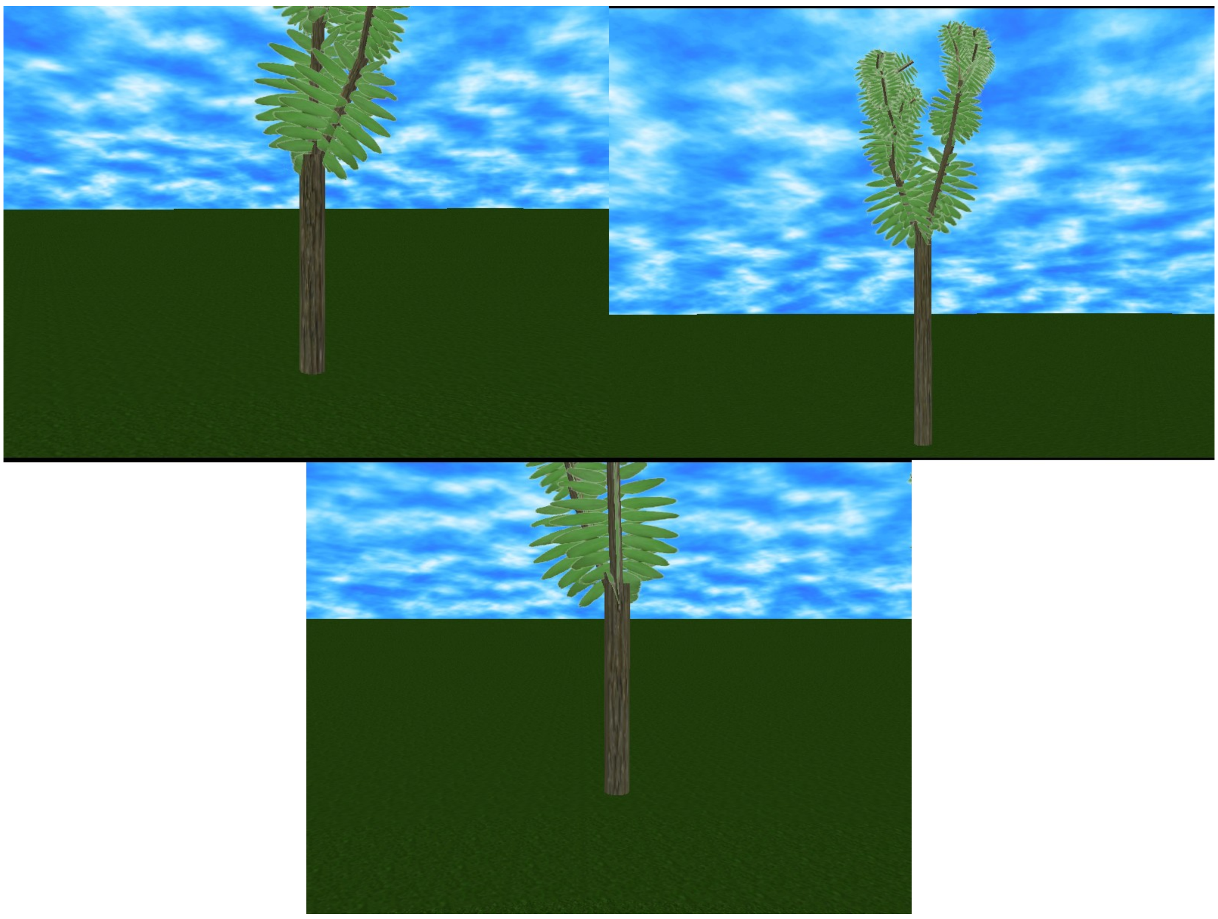


Illustration 26: Three examples that demonstrate overly long cylinders used in trees. The trunks of these trees appear extremely uniform and straight, unlike real trees.

For branches composed of many shorter segments, this is not such a problem, since the chain of segments approximates a real branch. However, for branches composed of fewer longer segments, the face that the branch is a collection of cylinders becomes much clearer, making the trees appear unrealistic.

There are a number of solutions to this problem. The most obvious one being to use many shorter segments in branches instead of fewer longer ones. Unfortunately, more segments makes the simulation more computationally expensive, and sometimes the best way to get the movement of a type of tree right is to use fewer segments.

For these reasons, there are a number of alternative approaches that could work. Firstly, using bump-mapping [21] on the branches would help to give them an appearance of texture and roughness, but would not help with the actual structure of the cylinder. To actually adjust the shape of the cylinder itself, free-form deformation [44], or displacement mapping [12], could be used to randomly alter the vertices of the cylinders on creation. This would reshape each cylinder uniquely to prevent them from all appearing the same. Alternatively, if a slight increase in graphical complexity is not a problem, multiple cylinders could be used to represent a single physics segment, increasing the visual appearance, without increasing the simulation complexity.

Ideally, we would have liked to have included the above enhancements in our system, but because the visual appearance of the trees was not the focus of our research, and because this problem was noted relatively late in the project, we chose not to add these features.

5.4.4 Miscellaneous Issues

There were also several other instances of unrealistic behaviour. For example, the gravity being too low by default, 'rubbery' behaviour in some of the trees, and the static nature of the leaves. These were all either known issues that were discussed in our design section, or easily fixable problems that require tweaking parameters of the simulation or L-system. Since these problems are easily fixed, or were already covered, we have chosen to not discuss them in this section.

5.4.5 Visual Heuristic Validation Conclusions

The conclusions we drew from the visual heuristic validation was that our trees mostly appear to be realistic, but have areas where their behaviour or visual appearance is noticeably unrealistic. Most of these problem areas are not major, and can be fixed by tweaking the L-system in question, or the parameters of the physics simulation.

There are a few cases though where the problem is slightly more deeply ingrained. In our case, these problems occurred due to limitations in the physics engine that we used, and inherent problems with discrete time-step simulations.

These problems are solvable, and we would have developed solutions, however time constraints and lack of access to the physics engine source code meant that we were unable to solve these problems within this research project.

We recommend that future implementations of our algorithm make use of fully open source physics engines and take these problems into account from the very beginning of development so that they can be dealt with appropriately.

6 Conclusion

To summarize our research, we designed and developed a system for enhancing procedural tree generation (specifically L-systems) so that the generated output incorporated physics components, allowing them to behave dynamically and realistically within their environment. Furthermore, the L-system itself is extended with symbols to control these physics components, and the parameters of the physics simulation.

In order to enhance the performance of our system, we used a number of optimizations and culling schemes. Some of these were well-known schemes provided by the pre-existing engines we used, and some were experimental ideas that we implemented.

We tested both the computational performance of our system (including the culling schemes we added), and the realism of the trees it created so that we could determine whether the system was practical and desirable in real-time applications.

Based on the results of our testing we can draw the following conclusions (which also answer our original research questions):

- It is possible to add the automatic creation of physics components to the procedural generation of trees (when using L-systems).
- Overall, the movements and dynamics in the produced trees are comparably realistic with trees in reality, and are more realistic than the trees found in current video games. There are some minor issues with our system, but these can be resolved Hence we conclude that the trees are acceptably realistic for use in practice.
- As it stands, our system is not suitable for real-time use with more than 50 trees being simulated, even if all optimizations and level-of-detail schemes are enabled. This performance limitation means that our system does not meet the performance requirements necessary for use in real-time systems. However, it is still applicable for off-line or pre-rendered applications which require a high level of realism.

In summary, our system produces promising results in terms of realism, but is not capable of fully realizing the results in real-time. It is entirely possible that specialized optimizations and/ or advanced level-of-detail schemes could change this situation and allow full real-time use with large numbers of trees. For this reason, we recommend that future research investigate such techniques and determine if they are capable of producing real-time performance from our algorithm.

6.1 Future Work

There are a number of areas that were raised during our research that are worthy of further investigation.

6.1.1 Physics Level-of-Detail

Firstly, an improved level-of-detail system for the physics engine would be advantageous. When doing large scale simulations involving many rigid bodies, such as a forest, and the accuracy of distant objects is not important, there are many possibilities for saving computation time, perhaps using schemes similar to those found in level-of-detail for graphics [10,22,30]. An investigation of such a system should also include a thorough examination of how it affects realism.

6.1.2 Simplified Physics Calculations

It appears possible to simplify the physics calculations even further if we assume that we will only be simulating trees. By combining this with the above level-of-detail suggestions, we believe that it should be possible to create a level-of-detail scheme that could be used to drastically increase the number of trees that can be simulated in real-time. This would be a valuable addition to the research in this paper.

6.1.3 Physics Components for Graphical Billboards

Related to the above point is the idea of adding some approximation of physics components to the billboards used in the billboarding level-of-detail scheme [10]. Billboarding is a widely used and effective method of handling level-of-detail for distant trees but, since the billboards are simply static images, they do not move and are not simulated by the physics engine. Hence distant (billboarded) trees would appear static while closer (non-billboarded) trees would be moving as simulated by the physics engine, creating a visual discontinuity that would negatively affect immersion and realism.

For this reason, we feel that research into techniques that would allow billboards to approximate the movements of trees that were being simulated would be a worthwhile endeavour. One approach that might be effective is using billboard clouds [10] instead of plain billboards and using cheap approximate simulations on the individual billboards that make up the cloud.

6.1.4 Generalized Cylinders

Another area for further research is rendering the tree branches and trunks using generalized cylinders [40]. In our system the limbs of the trees are rendered as a chain of cylinders, connected end-to-end. This is simple to implement and can provide an acceptable visual quality, but when the branches move significantly (such as in strong winds) the joints and rigid nature of the segments becomes visible. Generalized cylinders provide a way around this problem by rendering each branch/trunk as a parametric curve. In this way, the branches would appear to be a continuous curve, instead of a chain of rigid cylinders, and the movements of the branch would probably appear more sinuous. There may be problems using this technique with our system though, as it may be prohibitively expensive to do in real-time (even if it was, it could still be used in off-line applications), and generalized cylinders may not work well with dynamically moving branches (visual distortions could occur). We recommend future research, in order to establish whether generalized cylinders are viable with dynamic trees such as those in our system.

6.2 Recommended Improvements for Future Implementations

Since the implementation created during this research is only intended to be an experimental prototype for use as a proof-of-concept, and since we were operating under significant time pressures, there remain many areas for improvement.

In this section we will mention the most important ones that we feel should be included in any future implementations of the ideas contained in this research.

Firstly, a full scene graph [46] implementation would be a valuable addition. In our implementation, there is a flat list for each type of entity. A scene graph is a hierarchical tree data structure that groups entities spatially and can be used in a number of culling and level-of-detail schemes for significant performance gains [29]. For these reasons, we feel a full scene graph would enhance the program significantly.

We also recommend that an occlusion culling system [9,19,30] is added. In a forest situation, with a large number of trees, it is very likely that trees will occlude each other, providing the potential for significant performance gains if effective occlusion culling is enabled (it would tie in well with the scene graph suggestion above). We were unable to implement this due to the fact that OGRE does not support it, and adding the feature would have been an intensive endeavour beyond the scope of this project.

Better scripting support would also be a useful addition. Our scripting is somewhat limited, and was tacked on late in the development cycle. It was mostly oriented around moving the camera, and was used almost exclusively during performance testing. It did not have any support for creating entities or changing simulation parameters, which were the two things we desired the most. We strongly recommend planning for scripting support from the beginning, for the ease of use that it brings. This is of course doubly important in a video game situation, where any scripting should be integrated into the game's scripting engine.

Another enhancement that we recommend including is hardware acceleration of the physics simulation. As mentioned in the background chapter, there is a significant body of knowledge on running physics simulations on graphics cards. This technique is known to provide impressive speed benefits [20] and would doubtless increase performance of the system. Based on this, we advise that future implementations make use of this technology.

There are certainly other improvements and features that would be worthwhile additions to our software, but those outlined above are the ones that we believe would provide the most benefit in practical use.

7 References

1. Gimbal Angles and Gimbal Lock. http://history.nasa.gov/alsj/gimbals.html.
2. Abelson, H. and diSessa, A.A. Turtle Geometry. (1982).
3. Akgunduz, A., Banerjee, P., and Mehrotra, S. Smart collision information processing sensors for fast moving objects. *Smart Materials and Structures 11*, 1 (2002), 169-174.
4. Assarsson, U. and Moller, T. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools 5*, 1 (2000), 9–22.
5. Beaudoin, J. and Keyser, J. Simulation levels of detail for plant motion. *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association (2004), 297-304.
6. Behrendt, S., Colditz, C., Franzke, O., Kopf, J., and Deussen, O. Realistic real-time rendering of landscapes using billboard clouds. *Computer Graphics Forum 24*, 3 (2005), 507.
7. Boeing, A. and Braunl, T. Evaluation of real-time physics simulation systems. *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, ACM (2007), 281-288.
8. Brown, R., Cooper, L., and Pham, B. Visual attention-based polygon level of detail management. *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, ACM (2003), 55-ff.
9. Coorg, S. and Teller, S. Real-time occlusion culling for models with large occluders. *Proceedings of the 1997 symposium on Interactive 3D graphics*, ACM (1997), 83-ff.
10. D'ecoret, X., Durand, F., Sillion, F.X., and Dorsey, J. Billboard clouds for extreme model simplification. *ACM Trans.Graph. 22*, 3 (2003), 689-696.
11. Deussen, O., Hanrahan, P., Lintermann, B., Mvech, R., Pharr, M., and Prusinkiewicz, P. Realistic modeling and rendering of plant ecosystems. *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM (1998), 275-286.
12. Doggett, M. and Hirche, J. Adaptive view dependent tessellation of displacement maps. *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, ACM (2000), 59-66.
13. Eichhorst, P. and Savitch, W.J. Growth functions of stochastic Lindenmayer systems. *Information and Control 45*, 3 (1980), 217-228.
14. Eno, B. Generative music. *Imagination Conference*, (1996).
15. Faloutsos, P., Panne, M.V.D., and Terzopoulos, D. Composable controllers for physics-based character animation. *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM (2001), 251-260.

16. Goldiez, B., Rogers, R., and Woodard, P. Real-Time Visual Simulation on PCs. *IEEE Computer Graphics and Applications 19*, 1 (1999), 11-15.

17. Haevre, W.V., Fiore, F., Bekaert, P., and Reeth, F.V. A ray density estimation approach to take into account environment illumination in plant growth simulation. *SCCG '04: Proceedings of the 20th spring conference on Computer graphics*, ACM (2004), 121-131.

18. Herman, G.T. and Rozenberg, G. *Developmental Systems and Languages*. Elsevier Science Inc, New York, NY, USA, 1975.

19. Hudson, T., Manocha, D., Cohen, J., Lin, M., Hoff, K., and Zhang, H. Accelerated occlusion culling using shadow frusta. *Proceedings of the thirteenth annual symposium on Computational geometry*, ACM (1997), 1-10.

20. Joselli, M., Clua, E., Montenegro, A., Conci, A., and Pagliosa, P. A new physics engine with automatic process distribution between CPU-GPU. *Sandbox '08: Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*, ACM (2008), 149-156.

21. Kilgard, M.J. A practical and robust bump-mapping technique for today's GPUs. *Game Developers Conference 2000*, (2000).

22. Kumar, S., Manocha, D., Garrett, W., and Lin, M. Hierarchical back-face computation. *Proceedings of the eurographics workshop on Rendering techniques '96*, Springer-Verlag (1996), 235-ff.

23. Lam, Z. and King, S.A. Animation of tree development. *Image and Vision Computing New Zealand*, Citeseer (2003).

24. Lindenmayer, A. Mathematical models for cellular interactions in development, Parts I and II. *Journal of theoretical biology 18*, 3 (1968), 280-299.

25. Lindenmayer, A. Adding Continuous Components to L-Systems. *L Systems, Most of the papers were presented at a conference in Aarhus, Denmark*, Springer-Verlag (1974), 53-68.

26. Lindenmayer, A. and Rozenberg, G. Automata, Languages, Development. (1976).

27. Muller, M., Stam, J., James, D., and Thurey, N. Real time physics: class notes. *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, ACM (2008), 1-90.

28. Mvech, R. and Prusinkiewicz, P. Visual models of plants interacting with their environment. *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM (1996), 397-410.

29. Naef, M., Lamboray, E., Staadt, O., and Gross, M. The blue-c distributed scene graph. *Proceedings of the workshop on Virtual environments 2003*, ACM (2003), 125-133.

30. Nirenstein, S., Blake, E., and Gain, J. Exact from-region visibility culling. *Proceedings of the 13th Eurographics workshop on Rendering*, Eurographics Association (2002), 191-202.

31. Olsen, J. Realtime procedural terrain generation. *Department of Mathematics And Computer Science (IMADA) University of Southern Denmark*, (2004).

32. Parish, Y.I.H. and Muller, P. Procedural modeling of cities. *SIGGRAPH '01: Proceedings*

of the 28th annual conference on Computer graphics and interactive techniques, ACM (2001), 301-308.

33. Perlin, K. An image synthesizer. *SIGGRAPH Comput. Graph. 19*, 3 (1985), 287-296.
34. Popovic, Z. Controlling physics in realistic character animation. *Commun.ACM 43*, 7 (2000), 50-58.
35. Prusinkiewicz, P. Graphical applications of L-systems. *Proceedings on Graphics Interface '86/Vision Interface '86*, Canadian Information Processing Society (1986), 247-253.
36. Prusinkiewicz, P. Applications of L-systems to Computer Imagery. *Graph Grammars and their Application to Computer Science; Third International Workshop*, (1987), 534.
37. Prusinkiewicz, P., Hammel, M.S., and Mjolsness, E. Animation of plant development. *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM (1993), 351-360.
38. Prusinkiewicz, P., James, M., and Mvech, R. Synthetic topiary. *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, ACM (1994), 351-358.
39. Prusinkiewicz, P. and Lindenmayer, A. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc, New York, NY, USA, 1990.
40. Prusinkiewicz, P., M\ündermann, L., Karwowski, R., and Lane, B. The use of positional information in the modeling of plants. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, (2001), 289–300.
41. Roden, T. and Parberry, I. Clouds and stars: efficient real-time procedural sky rendering using 3D hardware. *ACE '05: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, ACM (2005), 434-437.
42. Sakaguchi, T. and Ohya, J. Modeling and animation of botanical trees for interactive virtual environments. *VRST '99: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM (1999), 139-146.
43. Salomaa, A. *Formal languages*. Academic Press Professional, Inc, San Diego, CA, USA, 1987.
44. Sederberg, T.W. and Parry, S.R. Free-form deformation of solid geometric models. *ACM Siggraph Computer Graphics 20*, 4 (1986), 151–160.
45. Seugling, A. and Rolin, M. *Evaluation of Physics Engines and Implementation of a Physics Module in a 3d-Authoring Tool*. 2006.
46. Sowizral, H. Scene graphs in the new millennium. *Computer Graphics and Applications, IEEE 20*, 1 (2000), 56-57.
47. Venter, J. and Hardy, A. Generating plants with gene expression programming. *AFRIGRAPH '07: Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa*, ACM (2007), 159-167.
48. Wong, J.C. and Datta, A. Animating real-time realistic movements in small plants. *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics*

and interactive techniques in Australasia and South East Asia, ACM (2004), 182-189.

49. Yokomori, T. Stochastic characterizations of EOL languages. *Information and Control 45*, 1 (1980), 26-33.

50. Zach, C., Mantler, S., and Karner, K. Time-critical rendering of discrete and continuous levels of detail. *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM (2002), 1-8.

51. Zamith, M., Clua, E., Conci, A., et al. A game loop architecture for the GPU used as a math coprocessor in real-time applications. *Comput.Entertain. 6*, 3 (2008), 1-19.

8 Appendices

8.1 Appendix A: Ethics

This appendix covers the ethical issues that were raised during the course of our research, and the approaches we used for dealing with them.

8.1.1 Licensing of Libraries

The largest ethical issue that arose during the implementation phase of our project was potential conflicts between the licenses of the external components that we used in our program. The software we created for our research made use of a number of open source packages and libraries (Ogre, Qt, Cmake, amongst others). Open source licenses vary in their exact details, but the common theme amongst them is that the source code of the program is freely available for scrutiny and modification. Comparisons of the licenses are available online[19]. The area where open source licenses vary is in how they allow the software to be integrated into, or used by, other programs. Some take a 'copyleft' approach and require that any software the program is used with is also open source (for example, the GNU General Public License, or GPL[20]), while others are far more permissive and do not restrict use so stringently (for example the BSD licenses[21]).

Although some of these (open source) licenses are compatible with each other (mostly provided that the software that is using them is open source), this is not always the case (for example the Apache license is incompatible with the GPL). We were expecting to release our implementation under one of the more widely compatible open source licenses, so we did not believe that this would be a problem for us.

The issue that arose was that PhysX (the proprietary physics engine that we used) is not open source, and is distributed under a proprietary license which allows free use for non-commercial projects (such as our research). This license is explicitly incompatible with certain open source licenses and is of questionable compatibility with others. For example, the license contains a clause disallowing integration with software distributed under 'viral' licenses which would change (or infect, using the virus metaphor) the license that PhysX is under (one such license is the GPL).

This issue was only discovered after we had begun the initial coding and were looking through the PhysX license. We noticed the clause stating that it was forbidden to use PhysX in any manner that would cause it to become subject to an open source license[22]. We immediately examined the licensing of the open source software that we were using to check if there was any conflicts. It turned out that there were none, but from that stage on, when we introduced any new libraries or programs into the implementation we carefully looked for any potential

19 http://developer.kde.org/documentation/licensing/licenses_summary.html
20 http://www.gnu.org/licenses/gpl-3.0.txt
21 http://www.opensource.org/licenses/bsd-license.php
22 http://developer.download.nvidia.com/PhysX/EULA/NVIDIA%20PhysX%20SDK%20EULA.pdf

conflicts in the license of that software. Luckily, none of the components that we wanted to use in our implementation were licensed in such a way as to cause a conflict with the PhysX license (by being distributed under the GPL, for example).

8.1.2 Licensing of Textures

Related to this licensing problem were the ethical issues surrounding the use of other individuals' content in the software that we developed. The main concern here was textures. Since our software involved displaying graphical representations of trees and landscapes, we needed textures to use on our generated models in order to produce acceptable visual results. Unfortunately we lacked the necessary knowledge and skill to create such textures ourselves. For this reason we searched for already existing textures that were available, and made use of those.

The ethical issue here was to make sure that we complied with the licensing requirements of the textures and were not ignoring any requirements in the license, or crossing any ethical lines. In the end, most of the textures that we used were distributed under a creative commons license[23] (a collection of licenses that allow non-commercial use of the licensed item provided that the original creator is credited), specifically, those creative commons licenses that allow derivative works, meaning that we could use them for the purposes of the project.

8.1.3 Releasing of Source Code

Another ethical issue raised during the course of the project was what we would do with the results of the project once it was concluded. This refers to the software and source code that we created during the project, since the research itself would be made publicly available by the Computer Science department of UCT (The University of Cape Town). As our source code was open source we would be obligated to make it freely available, but that still left open choices.

The main issue was whether we would simply release the source code and cease involvement, or whether we would continue to maintain, develop and release new versions of the software. Early on we decided that it was highly unlikely that we would have the time to maintain and develop such a project, which made us lean towards releasing without intention of further development.

Subsequently, it was pointed out to us that it could be considered unethical to release something and then abandon it (due to problems with the program, such as security holes, negatively affecting users). On balance though, we felt that it would be even more unethical to ignore the terms of the open source licenses that we agreed to use when we chose to use some of the software. However, in order to counter this potential ethical issue, we resolved to make it very clear that the software could be considered 'dead' (no longer actively developed) when we released it, which we believe will ethically abstain us from the problems around the lack of support.

23 http://creativecommons.org/about/licenses/

8.1.4 Commercialization

Linked to the issue of source code release, was the topic of commercialization of our results. Although the code itself is merely a proof-of-concept, and hence not suitable for commercialization, the ideas behind our project could be very useful in some commercial areas of computer graphics (such as video games).

There are, however, again some ethical issues with this. Considering that the research being conducted was always intended to be freely available, it could be seen as unethical to take the results and utilize them in a commercial venture where there would likely be a significant incentive to keep any improvements and further research secret (in order to gain/maintain a competitive edge in the market). This argument could be extended to suggest that the most 'ethical' course of action would be to release the results in the manner, and spirit, it was originally intended for: academic research.

For this reason, as well as the aforementioned lack of available time in the foreseeable future, we agreed that we would not pursue any commercialization of the research and would simply release it to the academic community.

8.1.5 User Testing

The final ethical issue that we needed to consider was the user testing. In any situation where subjects are involved in the testing of research it is important to respect them and follow all of the standard ethical practices.

In our case the testing was mostly very simple, and there were no ethical issues that would have needed clearance or special consideration. All of the normal ethical procedures were followed, such as not collecting any identifiable information from test users, and keeping their individual responses from being released outside the project itself. We also compensated the participants by paying them R20.00 for their time, and did our best to keep the length of testing process to less than one hour (per person).

8.2 Appendix B: User Testing Materials

This was the sheet that the user read first, to brief them about the test and it's procedure.

Physically Realistic Procedural Generation - User Study

Thank you for taking part in this user study. During the course of this experiment, you'll be shown a number of videos of computer simulated situations and be asked to answer questions based on how realistic you find the videos to be, and whether you think they would be realistic enough to pass in a video game/simulation. There are two sets of scenarios, one is simulated trees under a variety of forces, and the other is simulated building that are being demolished/destroyed.

For each of the two sets of scenarios, the following will proceed:

1. You will be shown a series of videos of real life examples of the scenario (involving either trees or buildings). You may watch these over as many times as you wish to in order to get a good idea of the movements/behaviour.

2. Then, you will be shown videos of a computer simulation of a similar scenario.

3. After you are finished watching you will be asked to answer a number of questions about how realistic you felt the simulated videos were in comparison to the real-life videos. You may refer to the simulated videos while answering.

4. Steps 1 – will be repeated a number of times, with different entities and situations each time. You will receive more detailed instructions for this.

5. After all of the sets of videos have been shown and the questions answered, you will be asked some general questions about all the videos.

6. You will then move onto buildings, if you have just done trees, or vice-versa.

After doing both the tree and building questions, you may, if you want to, interact with the simulation system and give additional commentary and discussion.

The simulated videos are rendered using computer graphics and thus won't have the detailed textures and colours that real life objects would. **The focus of this study is on the realism of how the trees and buildings move and interact with forces and other entities, not their visual appearance**. We ask that you bear this in mind when answering the questions.

Before the experiment begins, you will need to provide some basic information about you. This information will be kept completely confidential and only used in this study. No personally identifiable information will be included in our results. If you have any questions about the experiment, please feel free to ask them now.

This information sheet was provided to the user for them to use a reference when watching the videos. The description of the building scenarios was to do with the other research project who used the same test subjects as us, and ran simultaneously. It can be ignored, and is only included for the purposes of full disclosure.

Descriptions of the different tree simulation scenarios

1. Trees with no external forces acting on it except for gravity. This means that the only forces affecting the tree are the tension and springiness of its branches and trunk, and its weight under gravity.

2. Trees with winds of different strengths blowing at them from a single direction.

3. Large heavy boxes being thrown at trees.

4. Trees that have been cut at a point very low to the ground and are falling over as a result

5. A powerful explosion occuring in the middle of a group of trees.

6. A very strong tornado in the middle of a group of trees. (There are not any real-life videos of this, so please use your imagination to decide what this would look like in the real world)

7. A very strong attractive force (effectively a weak black hole) created in the middle of a group of trees. (There are not any real-life videos of this, so please use your imagination to decide what this would look like in the real world)

Descriptions of the different building simulation scenarios

1. Large 6 story tower being collapsed

2. Medium 3 story building being collapsed and hit with heavy projectiles

3. Small 2 story shed-like building being being collapsed and hit with heavy projectiles

This was the sheet used to collect information about the user.

User Information

• Age: ____________

• Gender: [F] [M] (circle one)

• What field(s) of study or work do you consider yourself to be in?
(e.g. Computer Science, Engineering, English, Commerce, etc)

Answer : ____

•During work and/or term time, how many hours a day on average do you spend playing video games? (video games include those on computers, consoles and hand-held devices such as the Nintendo DS)

Answer : ____

•If you did not have work and/or studies (e.g. were on holiday), how many hours a day on average do you think you would spend playing video games?

Answer : ____

•How many years/months have you been playing video games for?

Answer : ____

•Do you have any experience with video game programming, computer simulations and/or physics simulation? If so how much, and under what capacity?

Answer : ____

____

____

This sheet contained the instructions for the user doing the test, and was given to them to read before they began watching videos or answering questions.

Instructions for user for the trees section of the experiment

1. There will be 7 pages titled 'Tree Video Questionnaire (x / 7)'.

2. These pages will have a number between 1 and 7 (inclusive) handwritten in the top right corner.

3. In the folder that you are in on the computer that you are testing on there will be 7 sub-folders named '1' to '7'.

4. Open the folder corresponding to the handwritten number on the first sheet (e.g. if page 1/7 has a '3' handwritten in the top right corner, then open the folder labelled '3'). You will have a reference sheet which explains what the videos in each folder are meant to show. You may wish to refer to this to get an idea of what the videos are meant to show.

5. Within this folder there will be two sub-folders and two .m3u files.

6. Double click on real.m3u. This will open a movie player program with several movies queued up in the playlist. Watch these movies. Feel free to watch them more than once if you wish.

7. After you have watched all of those movies, double click on simulation.m3u. This will open a different set of videos. Watch these videos. Feel free to watch them as many times as you wish to.

8. Fill in the questions on the current page of the questionnaire (e.g. page 1/7 to continue the example from before).

9. Move onto the next page of the questionnaire. It will have a different number in the top right hand corner. Move up a level in the file browser and go into the folder whose name corresponds to the new number (e.g. If page 2/7 has '6' written in the top right corner, open the folder named '6'). Repeat the process of watching the 2 playlists in that folder and answering the questionnaire as per steps 5 – 8.

10. Continue this process until you've gone through all 7 of the pages and watched all 7 sets of movies. IMPORTANT NOTE: There are no real videos in folders number 6 and 7. These cover things that we could not find real-life simulations of. Please use your imagination to determine what you believe the situations described on the reference sheet would look like.

11. Finally, at the end of the questionnaire, there will be two pages of general questions. Please answer these questions with respect to all of the videos that you saw in the folders numbered 1 to 7.

12. You're done. Call one of us and let us know that you're finished.

The next three sheets were the questions that were asked in the questionnaire. There was a copy of the first sheet (individual quetions) for every scenario, while there was only one copy of the second two sheets (overall questions) answered per user.

Tree Video Questionnaire (1/7)

• On a scale of 1 to 7, how realistic did the events in the simulation video seem to you, compared to the events in the real life videos? (tick/cross one)

1.Totally unrealistic	[ ]
2.Unrealistic, with some elements of realism	[ ]
3.Unrealistic, but with a noticeable amount of realistic elements	[ ]
4.Equal amount of realistic and unrealistic elements	[ ]
5.Realistic, but with a noticeable amount of unrealistic elements	[ ]
6.Realistic, with some elements that were unrealistic	[ ]
7.Totally realistic	[ ]

• On a scale of 1 to 7, how realistic would you consider the video in terms of what you would expect in a video game? (tick/cross one)

1.Totally unrealistic	[ ]
2.Unrealistic, with some elements of realism	[ ]
3.Unrealistic, but with a noticeable amount of realistic elements	[ ]
4.Equal amount of realistic and unrealistic elements	[ ]
5.Realistic, but with a noticeable amount of unrealistic elements	[ ]
6.Realistic, with some elements that were unrealistic	[ ]
7.Totally realistic	[ ]

• a) Where there any features or elements that you noticed in the real-life videos that were not in the simulation videos?

Answer : ____

____

____

• b) Where there any features or elements that you noticed in the simulation videos videos that were not in the real-life videos?

Answer : ____

____

____

• Are there any other comments or observations you wish to give? (Comments on the realism of the simulation, unrealistic moments, etc.) **Please answer on back of page.**

Tree Videos General Questions (1/2)

• What was your overall impression of the realism of the simulations shown in the videos (based on all the videos shown)?
Answer on a scale of 1 to 10 where:

- •1 means totally unrealistic,
- •5 means somewhat realistic but had noticeably unrealistic elements
- •10 means completely realistic

Answer: _______

Comments: ____

____

____

____

• Do you feel the tree simulation videos you were shown are an improvement over existing tree realism in current video games?
Answer on a scale of 1 to 10 where:

- •1 means negative improvement (i.e. current games are more realistic),
- •5 means no improvement, and
- •10 means a massive improvement

Answer: __________

Comments: ____

____

____

____

Tree Videos General Questions (2/2)

•Are there any suggestions that you could give that you think would make the simulations appear more realistic?

Answer: ______________________________

• Any other comments or observations?

8.3 Appendix C: Raw Testing Data

8.3.1 Performance Testing

The following tables contain the raw data collected during our performance testing. The numbers in the cells are the average times to update the graphics or physics for a single run, in microseconds. The text at the top of the table indicates the parameters of that run, and the column headings indicate the number of trees that were present in that run.

Base Case											
Graphics						Physics					
0	50	100	150	200	250	0	50	100	150	200	250
439.5	17749.1	31075.8	60624.4	106002	101458	160.97	6927.71	13534.1	21927.7	31024.1	39056.2
449.55	16216.6	41710.1	73250	81156.5	111807	220.88	6586.35	15722.9	22208.8	32510	37751
487.44	18851.7	44289.2	57462.2	89344.2	100348	100.4	7951.81	14277.1	23714.9	31807.2	39236.9
421.94	21708.5	35409.1	64046.5	82044.2	102794	361.45	7289.16	14779.1	22730.9	31867.5	40562.2
443.25	19238.2	48140.6	54538.9	74933.6	143512	80.32	7148.59	15642.6	22269.1	28935.7	40843.4
394.84	21715.1	34139.3	68148.5	105902	89263.1	160.64	7690.76	13072.3	23554.2	32048.2	37008
408.96	17267.5	33237.3	60864	68729.9	115337	100.6	6847.39	15301.2	20903.6	30502	42911.6
983.38	15635.7	48571	49457.4	83802.5	125924	301.21	6807.23	16726.9	23975.9	31506	36747
440.04	19398.6	30822.6	57619.3	102722	99420.8	100.4	7349.4	15000	23634.5	32249	38674.7
451.23	20413.3	41071.1	71292.2	86805	111768	180.72	8534.14	16004	22208.8	32931.7	39698.8
547.02	16239.9	49148.5	59626.2	85350.9	94491.6	361.45	7008.03	14558.2	23935.7	32068.3	36907.6
396.89	21605.3	34967.5	81556.1	77847.6	99622	180.72	7389.56	14156.6	26144.6	33152.6	40843.4
488.67	18836.63	39381.84	63207.14	87053.37	107978.79	192.48	7294.18	14897.92	23100.73	31716.86	39186.73

Backface Culling Off											
Graphics						Physics					
0	50	100	150	200	250	0	50	100	150	200	250
412.83	19569.6	31807.3	70197.7	107341	108386	201.21	6104.42	14618.5	24357.4	31064.3	42329.3
419.39	15742.1	43879.8	49582.3	87761.7	135660	160.64	5722.89	15241	25160.6	34196.8	38714.9
390.36	18962.8	52466.1	61658.8	90939.5	103739	301.21	8092.37	15622.5	23755	32751	38975.9
415.75	18782.8	32928.6	76474.5	82725.6	112765	160.64	7249	13052.2	20963.9	32148.6	44236.9
432.88	16910.4	51440.9	63763.6	77894.7	103913	180.72	7228.92	15702.8	23413.7	29919.7	40040.2
918.22	19132.3	32578.2	71240	107266	109632	441.77	7429.72	14176.7	23313.3	30261	39176.7
429.8	16697	32101.5	64890.2	65833.1	109895	261.57	8092.37	13815.3	23232.9	28915.7	40060.2
407.24	15147.3	40237.9	76548.1	85971.9	117238	220.88	6807.23	14638.6	22710.8	30622.5	40923.7
421.02	18807.5	33241	60420.3	100785	102297	100.4	7369.48	15060.2	24257	32148.6	40120.5
425.72	21579.7	39997.1	57997.9	81472.8	113056	160.64	7068.27	15401.6	24116.5	31586.3	41807.2
577.97	17391	56045.5	55632	86441.5	145160	80.32	6967.87	16204.8	24959.8	31365.5	42168.7
438.83	18776.6	38968.5	63116.6	77359.2	97268.1	220.88	7570.28	15763.1	22208.8	32028.1	43614.5
474.17	18124.93	40474.37	64293.5	87649.33	113250.76	207.57	7141.9	14941.44	23537.48	31417.34	41014.06

Frustum Culling Off											
Graphics						Physics					
0	50	100	150	200	250	0	50	100	150	200	250
425.7	16753.8	30394.3	63274.3	96638.3	108540	200.8	7851.41	12449.8	22650.6	34076.3	36586.3
1035.54	17432.4	43992.7	76760.4	84666.7	120539	180.72	6084.34	13293.2	27108.4	25140.6	42008
607.58	18458.5	43641.1	59310.1	83697.3	108657	160.64	6947.79	15803.2	20722.9	30642.6	43895.6
435.42	19080.6	36582.1	64741.5	77409.8	109968	140.56	8112.45	12891.6	22730.9	31345.4	51967.9
478.11	17038.2	41710.4	55473.8	76733.6	146687	200.8	6847.39	15662.7	26405.6	35702.8	40963.9
395.83	18248.4	34266.7	63550.5	107845	97061.9	100.6	6204.82	14236.9	24718.9	28232.9	39779.1
381.99	17677.4	34028.2	61589.2	109109	96871	120.72	7570.28	14658.6	26967.9	34538.2	40220.9
481	18903.3	46403.9	59303.3	74257.4	149681	160.64	6164.66	13915.7	26124.5	38052.2	40180.7
402.79	21825.6	36193.2	58239.4	79463.3	106057	160.64	8614.46	13333.3	23453.8	37289.2	47048.2
494.45	18440.5	42474	56104.6	88895.2	101666	120.48	8313.25	16285.1	20722.9	32530.1	40180.7
828.63	17886.9	39307.9	73674.4	82857.9	126813	200.8	6445.78	13273.1	26526.1	27690.8	42590.4
424.81	20351.3	29557.9	63485.1	104637	108370	200.8	7008.03	12188.8	21887.6	35481.9	37228.9
532.65	18508.08	38212.7	62958.88	88850.88	115075.91	162.35	7180.39	13999.33	24168.34	32560.25	41887.55

Distance Culling Off											
Graphics						Physics					
0	50	100	150	200	250	0	50	100	150	200	250
557.04	23428.4	54123.4	98464.3	104044	153317	828.18	19763.62	47961.83	78086.21	94566.54	124191.52
517.39	23804.5	68109.4	82209.1	119113	136550	691.22	22650.88	51601.18	66102.44	105566.23	122656.06
512.47	38179.9	49304.7	96189.9	107356	163080	611.51	24964.41	46714.97	78276.81	113007.5	122763.27
619.03	30383.4	67621.9	78513.6	116329	178309	869.65	21721.81	44742.02	74040.95	89150.3	125755.46
510.18	24828.3	53858.9	72134.7	142518	138344	541.73	20950.85	47357.9	67148.35	104650.23	127362.8
578.4	26726.4	59170.4	103190	107191	150662	840.92	19615.39	47397.85	57995.66	98870.28	124373.14
522.71	27266.4	55757.2	95168.6	130906	140103	854.62	18866.95	50369.67	68789.73	95944.24	119034.92
1234.14	26939.2	66696.4	81584.6	117851	175639	815.85	18019.42	40354.26	66884.75	95291.36	124685.35
514.66	34650.4	56633	90917.6	113858	157242	409.39	21817.48	44103.81	86544.08	104314.12	124024.05
562.57	31555.6	69989.3	84007.4	128933	140156	560.09	21053.71	44846.89	70349.08	99520.98	125080.67
1778.92	28488.1	53742.5	109990	111754	168029	414.94	24333.13	48173.7	72420.59	103179.59	121320.26
479.05	21364.9	56414.9	75073.8	136826	143249	762.78	21474.48	47791.67	65182.63	108820.89	124605.3
698.88	28134.63	59285.17	88953.63	119723.25	153723.33	683.41	21269.34	46784.64	70985.11	101073.52	123821.07

All Optimizations Off											
Graphics						Physics					
0	50	100	150	200	250	0	50	100	150	200	250
820.12	20583.8	54410.1	100327	114566	158259	583.5	20863.5	51425.7	65883.5	104980	123012
1552.94	26854.9	66116.3	81247.5	126954	133698	441.77	22911.6	43554.2	75441.8	103153	132851
1479.05	36961.9	57393.9	102536	98015.7	143024	1084.34	20763.1	47851.4	72490	104317	138173
562.05	29086.7	64305.3	83712.9	113522	177078	461.85	23815.3	46144.6	72128.5	100964	129799
537.11	24193.9	53303.9	89459.1	141496	149845	1124.5	24638.6	56947.8	70100.4	97008	122510
450.15	27112.8	61370.1	102097	107971	156791	502.01	23855.4	47469.9	79397.6	98614.5	123032
456.7	26156.5	57143	112380	107286	153459	220.88	23775.1	50622.5	71144.6	99156.6	122871
1545.33	30392.3	60953.8	82307.5	134776	137352	401.61	19477.9	51044.2	76646.6	100884	137229
579.1	26902.8	56138.9	77320.4	116553	182106	582.33	23915.7	50582.3	77831.3	98373.5	127309
1512.23	30325.2	53147.6	86616.3	110073	156175	1224.9	24899.6	49417.7	83433.7	92469.9	120643
559.96	28688.9	63662.5	84472.3	129263	141942	1325.3	23815.3	46245	73072.3	104357	128273
555.03	21743.7	58890.1	92583	109999	151250	764.59	19979.9	48012	72409.6	108534	126807
884.15	27416.95	58902.96	91254.92	117539.56	153414.92	726.46	22725.92	49109.78	74164.99	101067.63	127709.08

8.3.2 User Testing

These two tables are all of the quantitative data we gathered during the user testing. We have only included the results of the test subjects whose results were used in our analysis. We have left out the pilot test subject and the one test that was disqualified due to a power failure.

This first table is the user information that we collected. Note that the entries in the three rightmost columns are simply the answers that the user wrote on the questionnaire, copied as exactly as was possible. The final question (experience in creating computer games and simulations) is not included as these answers were quite lengthy.

User Number	Age	Gender	Field of Study	Hours playing games	Hours would play	Years/Months playing
1	21	M	Computer Science	Alot (35 hours a week +-)	48 hours a week +-	13 years +-
2	22	F	Computer Science	1 hour	3 hours	4 years
3	22	M	Computer Science and Psychology	45 mins – an hour	two to three hours	13 years
4	22	F	Humanities – Philosophy	0	0	Since about eleven years
5	21	M	Computer Science	½ an hour	2 hours	7 years
6	22	M	Computer Science	0.5 hours	4 hours	10 years
7	21	M	Computer Science	0.5	3	10
8	21	M	Actuarial Science	0	20 min	4 years
9	22	F	Humanities – Psych	About 1 hour a week	Probably only the same as above	At least 15 years
10	21	M	Computer Science	4	8	8 years
11	22	M	Computer Science	1 – 2 hours	1 – 2 hours	18 years
12	22	M	Computer Science	2 – 3	4 – 5	+- 10
13	21	M	Computer Science	1 – 2	4	10
14	24	F	Computer Science	zero	zero	I played as a kid so probably 10 years
15	24	M	Computer Science	1 hour	5 hours	20 years
16	19	F	Computer Science	half an hour to an hour	about three	7 years
17	19	M	Computer Science	2 hrs	8 hrs	at least 12
18	22	M	Computer Science	2	3	14 years
19	22	F	Computer Science	0 hours/day	1 hour/day	+- 12 years
20	24	M	Computer Science	1 hour a week	3 – 6 hours/day	19

The next table contains the raw answers that the users gave for the questions. The column labelled 'First' indicates whether the user did our testing first or the testing of the other project, that we were sharing users with, first. Half of the users did our testing first.

An entry of 'N/A' means the user chose not to answer the question because they did not believe they were qualified to.

		Individual Questions														Overall Questions	
Number	First	Q1A	Q1B	Q2A	Q2B	Q3A	Q3B	Q4A	Q4B	Q5A	Q5B	Q6A	Q6B	Q7A	Q7B	Q1	Q2
1	N	7	7	7	7	6	7	6	7	6	6	6	6	6	6	9	5
2	Y	7	7	6	7	6	7	7	7	5	6	6	7	7	7	9	8
3	N	5	6	4	5	4	5	5	6	6	7	6	6	7	7	6	8
4	Y	6	7	6	6	6	6	5	6	5	5	2	3	7	7	8	N/A
5	N	7	7	5	5	7	7	7	7	6	6	7	7	7	7	9	5
6	Y	5	6	6	6	5	5	7	7	5	5	7	6	6	6	7	6
7	N	5	5	5	6	5	5	6	7	4	5	6	7	4	5	8	9
8	Y	4	4	6	6	3	3	6	7	2	3	3	5	3	4	7	N/A
9	N	6	6	6	7	5	5	6	6	5	6	5	6	4	5	8	5
10	Y	4	5	5	6	5	6	5	5	5	5	6	7	5	5	8	6
11	N	7	7	5	6	6	6	6	6	3	4	5	6	6	6	7.5	9
12	Y	4	4	3	4	2	3	3	5	2	3	3	3	4	6	6	5
13	N	4	5	5	6	5	5	4	4	2	4	5	6	4	5	5	10
14	Y	5	6	3	3	7	7	6	6	3	5	6	7	7	7	8	N/A
15	N	6	6	7	7	7	7	6	6	5	5	5	5	4	4	7	6
16	Y	6	7	6	7	5	6	5	6	5	6	6	6	6	7	8	9
17	N	7	7	5	6	4	5	7	7	6	7	6	7	5	6	7	9
18	Y	5	5	3	5	5	5	6	7	3	5	5	6	4	5	7	7
19	N	5	6	6	6	6	6	2	6	3	5	6	6	6	6	8	10
20	Y	6	6	3	3	3	3	5	5	3	4	2	2	4	4	5	6