Automatic Addition of Physics Components to Procedural Content

Richard Baxter* University of Cape Town
Zacharia Crumley† University of Cape Town
Rudolph Neeser‡ University of Cape Town
James Gain§ University of Cape Town

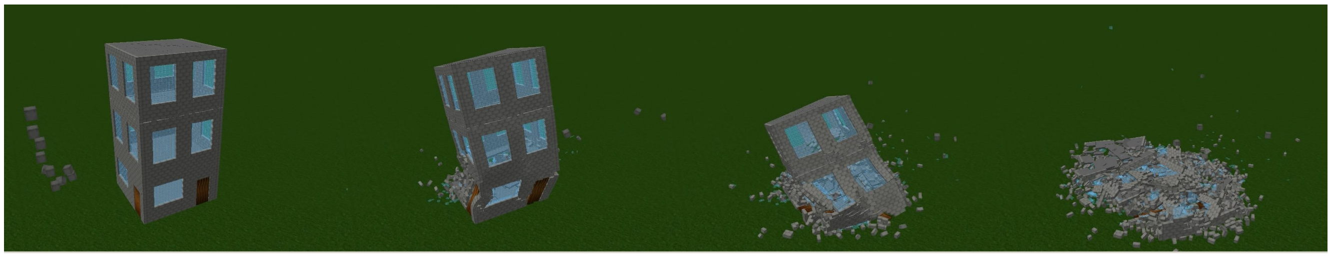


Figure 1: *Four frames of an animation produced by our system. The building is procedurally generated and ready to undergo physical animation. It collapses here after being struck by large blocks (seen in far left).*

Abstract

While the field of procedural content generation is growing, there has been somewhat less work on developing procedural methods to animate these models. We present a technique for generating procedural models of trees and buildings *via* formal grammars (*L-Systems* and *wall grammars*) that are ready to be animated using physical simulation. The grammars and their interpretations are augmented to provide direct control over the physical animation, by, for example, specifying object mass and the joint stiffness. Example animations produced by our system include trees swaying in a gentle wind or being rocked by a gale, and buildings collapsing, imploding or exploding. In user testing, we had test subjects ($n = 20$) compare our animations with video of trees and buildings undergoing similar effects, as well as with animations in games that they have played. Results show that our animations appear physically accurate with a few minor instances of unrealistic behaviour. Users considered the animations to be more realistic than those used in current video games.

Keywords: Procedural Generation, Physics, Dynamic Animation, L-Systems, Wall Grammars, Descriptive Grammars, Shape Grammars

1 Introduction

The procedural generation of content — that is, the production of models, textures, animation, and so on, by algorithmic means — is of increasing importance to many industries, from CGI and computer games to reverse engineering and urban planning [Watson et al. 2008].

An important aspect of procedural content generation is the animation of this content: models are often required to show behaviour appropriate to their type (*e.g.,* people should walk; trees should sway in the wind), while this behaviour should be modified by the physical properties of the model (*e.g.,* people have differing strides depending on, say, their height; plants may be woody or herbaceous, affecting their flexibility).

We present a simple method for automatically augmenting models produced by formal grammar methods so that they can be animated using physical dynamics. Unlike other animation methods, such as skinning, dynamics has little reliance on an artist to produce an appropriate animation sequence, making it suitable for inclusion in automatic content production. We demonstrate how to do this for two broad categories of formal grammars: those typically used for producing plants (exemplified here with an *L-System* method, which is a class of grammar that operates directly on *symbols* as opposed to *shape* or *geometric objects*; this is called a *descriptive* grammar) and those typically used for producing buildings (implemented here using *wall grammars*, which operates directly on *walls* rather than symbols, and is in essence a type of *shape grammar*). These two techniques are chosen because they exemplify much of the procedural model generation work in the current literature: the generation of *floral ecosystems* (such as in the work of Prusinkiewicz, *e.g.,* [Prusinkiewicz and Lindenmayer 1990; Deussen et al. 1998; Mvech and Prusinkiewicz 1996]) and *urban landscapes* (such as the work of Müller and Wonka, *e.g.,* [Parish and Müller 2001; Wonka et al. 2003]), both of which rely strongly on formal grammars. While we focus on real-time[1] applications, the methods presented here are easily applied to offline rendering, as well as to procedural methods not based on formal grammars.

Grammars come in two types, those described as *descriptive* — which operate on symbols, and hence have a separate *interpretation* step to create geometry from those symbols — and *shape grammars*, which operate directly on shape / geometry, and have no interpretation step. There is, unfortunately, no straightforward way to unify how physical components are added to the two grammar families, and so we provide solutions for both.

We extend the *L-System* formalism with symbols for controlling properties of a physics simulation (such as an object's mass, or the constraints on a joint). No limitations are placed on the geometric output. For each geometric object undergoing simulation, the interpretation of the *L-System* symbols also includes output meant for the physics simulation. As the simulation modifies these objects, their movement is mapped onto the corresponding geometry.

For shape grammars, we show how the geometry can be subdivided to produce the material components (brick, wood, *etc.*) constituting the object. By keeping track of the surrounding geometric context

*e-mail: richard.jonathan.baxter@gmail.com
†e-mail: zacharia.crumley@gmail.com
‡e-mail: rudy@cs.uct.ac.za
§e-mail: jgain@cs.uct.ac.za

[1]We define *real-time* animation as having frame rates in excess of 30 frames per second.

as the geometry is subdivided we can also include material components, such as bricks, which may overlap the various larger shapes used to create the bulk of the model. Again, this process should be applicable to other methods, such as split grammars.

The output of these grammars are models that are ready to be animated using a dynamics engine. This can be used to trivially produce animations of trees gently swaying, blowing violently, or merely sagging under gravity. Buildings can easily be made to collapse, or to react to external forces such as a wrecking ball, explosives, vehicular accident, and so on.

An important aspect of our work concerns user testing: we have presented our animated models to users ($n = 20$) and allowed them to compare the models' behaviour to videos of actual trees and collapsing buildings. We also asked the users to rate the realism of our animations in relation to games. The results show that the users find the animations realistic, although some unrealistic elements remain (such as there being a lack of dust clouds when a building collapses), most of this appears to be easily correctable. The users also find our animations to be an improvement on those seen in current games. Apart from this, we report on some aspects of our performance testing to see how close to real-time performance we have come. We did not make use of hardware accelerated physics simulation and relied solely on a software implementation. This does mean that our obtained performance is less than what we wanted, and we did not manage to meet our real-time needs.

The paper begins by covering some background in physics simulation (§ 2.1) and grammar-based procedural methods for flora and building generation (§ 2.2). Section 3 presents the implementation details related to augmenting these formal systems to manage the physics simulation. Next, we cover the experimental methods used for the testing (§ 4), and present and discuss the obtained results (§ 5). Finally, we conclude (§ 6).

2 Background

2.1 Physical Dynamics

Computer simulated physics is widely applied in films, games and virtual environments, and has been an area of intense ongoing research for decades, beginning with simple rigid-body animations of objects obeying Newtonian mechanics [Badler 1982; Girard and Maciejewski 1985; Wilhelms 1987]. While the broad principles of motion dynamics have been well understood for some time there is ongoing work in adapting them to specific circumstances, such as liquids [Wang et al. 2009], cloth [Kaldor et al. 2008], hair [Selle et al. 2008], and deformable bodies [Nesme et al. 2009], where requirements for visual realism differ. A further challenge lies in attempting to make such simulations real-time, which allows improvement in the realism, and hence immersion, of interactive media.

In this paper, we focus specifically on jointed rigid bodies because these forms of simulation are relatively mature, as evidenced by the availability of a number of stable physics engines [2] [3] [4], and capable of supporting both tree and building dynamics. For the interested reader, Boeing and Bräunl [2007] and Seugling and Rölin [2006] provide comparisons of existing physics engines.

In jointed rigid-body dynamics a number of rigid segments (bones) can be connected with rotational constraints (joints) to form a hierarchy (or skeleton). Physical dynamics can then be applied to the skeleton so that it moves in a believable fashion under various forces, such as gravity and wind [Faloutsos et al. 2001; Popovic 2000]. A typical example of this is the 'rag-doll' physics found in video games (*e.g.,* Unreal Tournament 2003), where characters 'realistically' fall, roll and collide with their environment.

2.2 Grammar-based Procedural Methods

Grammar-based procedural methods consist of a set of substitution rules that operate on an initial string of symbols to produce a final string. This string is then interpreted in some way to produce the wanted output. *L-Systems* (*Lindenmayer System*) are an early example of a grammar-based modelling and simulation system, developed in 1968 by Aristid Lindenmayer as a method for simulating the growth of simple multi-cellular organisms [Lindenmayer 1968]. Since then they have been employed in a wide range of applications, such as the modelling of cities and road networks [Parish and Müller 2001], plants, trees and ecosystems [Prusinkiewicz and Lindenmayer 1990].

To interpret the symbols of an *L-System* as a tree, Prusinkiewicz proposed an approach similar to the turtle graphics used in LOGO [Abelson and diSessa 1982; Prusinkiewicz 1986; Prusinkiewicz 1987]: the generated strings become instructions to a plotter pen (the eponymous *turtle* in turtle graphics) drawing the basic skeleton of the tree. Extensions [Prusinkiewicz and Lindenmayer 1990] update this process to produce geometric models rather than merely line drawings.

The basic *L-System* formalism has been expanded to produce open *L-Systems* [Mvech and Prusinkiewicz 1996], differential *L-Systems* [Prusinkiewicz et al. 1993] and so on, allowing *L-Systems* to query the model as it is being produced, to query the environment in which the model is being produced, and to understand continuous changes in the model for animations of growth. This range of subtypes means *L-Systems* have a correspondingly wide range of uses, as can be seen in their extensive usage in procedural generation, such as by Deussen *et al.* [1998] and Parish and Müller [2001].

Additionally, Stiny and Gips [Stiny and Gips 1972] presented a grammar-based approach in which the symbols are replaced with geometric shapes. Rather than having an initial string of symbols, the system has an initial shape, and rather than having a separate interpretation step, each substitution *directly* affects the geometry of the initial shape. While shape grammars have been heavily used in architecture, in the computer graphics literature they have been replaced with *split grammars*.

The *split grammar* [Wonka et al. 2003; Müller et al. 2006] is a shape grammar that operates on geometry that is iteratively split, allowing a base-shape to be refined into a building. For example, a cuboidal block can be split vertically into floors, each floor can be further split to form windows, doors and other building structures. A wide variety of buildings and architecture can be constructed using this approach [Wonka et al. 2003; Müller et al. 2006].

The *wall grammar* is a two-dimensional split grammar [Larive and Gaildrat 2006] that operates solely on building façades (2D faces). These façades are joined together with a roof to form a final 3D building model. Similar to split grammars, wall grammars split 2D rectangles into individual floors, then divide these floors into windows, doors, and so on, to create a single façade [Larive and Gaildrat 2006].

Using a basic floor-plan, the wall grammar is able to create a wide variety of building shapes and styles. Whilst not as varied as the 3D split grammar, it is far simpler to implement and, according to Larive and Gaildrat [2006], simpler to use.

[2]*Bullet Physics*: http://www.bulletphysics.com/
[3]*PhysX*: http://developer.nvidia.com/object/physx.html
[4]*Dynamics Engine*: http://www.ode.org/

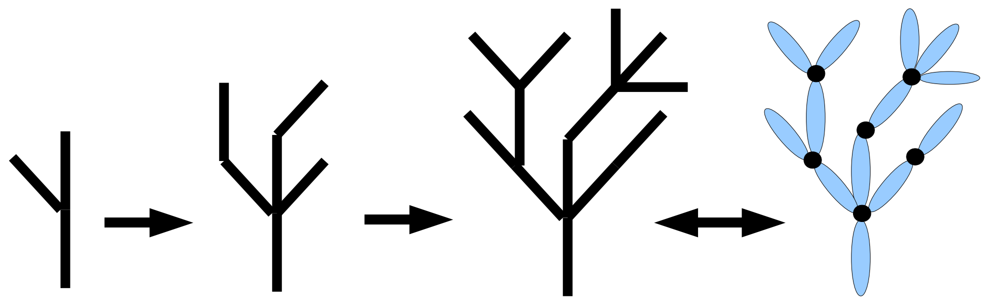


Figure 2: *An example of an* L-System *being used to create a simple tree over three iterations, along with the physics skeleton generated for the final tree.*

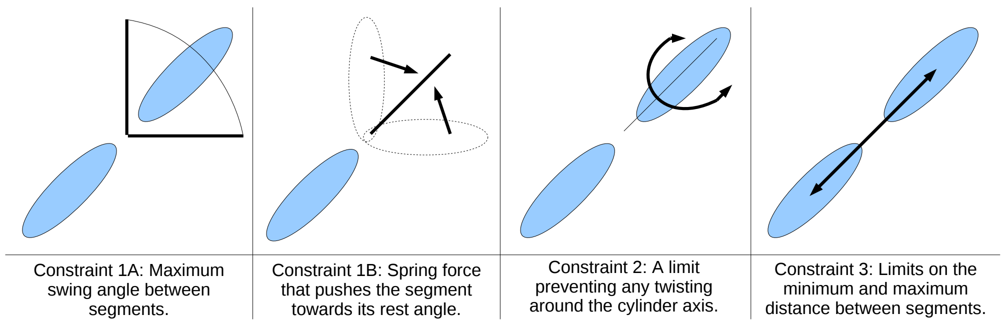

Constraint 1A: Maximum swing angle between segments.
Constraint 1B: Spring force that pushes the segment towards its rest angle.
Constraint 2: A limit preventing any twisting around the cylinder axis.
Constraint 3: Limits on the minimum and maximum distance between segments.

Figure 3: *A diagram explaining the four constraints used in the joints between adjacent physics bones.*

2.3 Related Work

There has been little research into extending procedural generation grammars for physics, particularly with respect to buildings.

Sakaguchi and Ohya [1999] show how to animate models of trees by dividing them into rigid segments and calculating their movements with a highly realistic two phase simulation. The first phase calculates the movement of individual branch segments, and the second phase integrates these individual movements together into the overall tree simulation. While this is similar to our work — we also use a branch segment approach — Sakaguchi and Ohya generate their trees from captured image data rather than procedurally. Further, they aim to produce a highly realistic physics model, and real-time interaction is not a priority of their work. In spite of this, parts of their research inform the simulation methods employed in our system.

Wong and Datta [2004] demonstrate similar work for animating plants. Their technique is intended to be usable in real-time and follows a similar approach to Sakaguchi's by dividing the plant into rigid segments and placing constraints on the movement between adjacent segments. Their research is aimed at animating low numbers of small, soft plants, and places a heavy emphasis on leaf animation. In contrast, we are more interested in large numbers of fully grown trees. Additionally, they use pre-created models with randomised branch placement, as opposed to a grammar-based approach.

There has been work on animating the *growth* of models produced using *L-Systems*, since *L-Systems* lend themselves well to this. Substitution rules are inherently discrete, changing the string (and hence its interpreted shape) in steps. Prusinkiewicz *et al.* [1993] present a way of using differential equations to specify how the model changes continuously between these steps. This differs from our work in that we are animating the models *after* production; Prusinkiewicz *et al.* [1993] are concerned with the animation (and simulation) of the *production* itself.

While buildings are often destroyed in film, or even in games, we have found no published work on how to procedurally generate buildings to which external forces can trivially be applied and animated using physical simulation.

3 Implementation

We implemented tree and building generation using two separate grammar-based systems: *L-Systems* for the trees, and *wall grammars* for the buildings. Standard rules and interpretations of these grammars (see below) were extended to provide control over the model's motion dynamics. The interpretation of the string produced by the *L-System* generates not only the geometric model of the object, but a separate skeleton suitable for use in the physics simulation. The wall-grammar, since it operates directly on geometry, has no modified interpretation, but merely additional physical properties belonging to the shapes themselves, and additional substitution rules for breaking the shapes into their constituent components. All the physical simulation occurs *after* the creation of the final geometric output. The physical simulation plays no part in the application of the rewrite rules in each grammar, although if, say, open *L-Systems* [Mvech and Prusinkiewicz 1996] were implemented, then the physical simulation itself could guide plant development. While we wrote our own grammar systems, the physical simulation was executed using off-the-shelf software (namely *PhysX*).

More details of the design and implementation of our system can be found in Baxter [2009] and Crumley [2009].

Hereafter, we discuss first the creation of trees, and then buildings.

3.1 Tree Creation: the Descriptive Grammar Approach

We employ a fairly simple context-sensitive, parametric, stochastic *L-System* [Prusinkiewicz and Lindenmayer 1990], although the method described here should trivially apply to any other *L-System*. Trees are generated in three stages: a string representing the tree is generated using an *L-System* grammar; this string is interpreted to produce the tree geometry; and the same string is interpreted again to produce a skeleton for the physics simulation. This skeleton is associated with the tree geometry.

The interpretation we use to produce the tree geometry is similar to the turtle graphics approach proposed by Prusinkiewicz, an excellent overview of which can be found in the *Algorithmic Beauty of Plants* [Prusinkiewicz and Lindenmayer 1990]. A simple visual example of our algorithm generating a tree can be seen in Figure 2.

Importantly, the 'turtle' is given instructions defining branch density, thickness, texture, and so on. These parameters can be modified *via* symbols in the *L-System* and are used at later stages in the tree creation process to enhance the range and diversity of the output. We represent each branch with a textured cylinder.

The physics skeleton is created by defining a rigid physics 'bone' along each segment of the tree branch; these bones are connected together with appropriate joints. This is similar to the approach used by Sakaguchi and Ohya [1999], but has been significantly modified for use with a physics engine. The bones define the shapes used in the physics simulation, and the joints define the forces and constraints that occur where two or more shapes join. We represent the bones using capsules (3D ellipsoids). The constraints placed on the joints are discussed further below, while the symbols controlling the physical properties of the model are described in Table 1.

In practice, it is necessary to make a physics capsule slightly shorter than the length of the branch. If it spans the full length, adjacent capsules overlap, causing undesirable interactions between components of the tree, and ultimately resulting in unwanted movement. We recommend that the physics bones be between 80% and 95% of the actual length of a branch segment. The visual representation of the branch, on the other hand, should remain at the full length.

We create leaves as simple billboard textures, using alpha channels to allow transparency. Two perpendicular billboards are attached to each branch segment. While this is a simple method, the physical simulation should easily support any leaf generation method, even those that produce leaf geometry.

A physics bones is connected to those that occur both before and after it in the branch. Since the final segment in a branch has no successor, it is a terminating bone. Also, the first segment in a branch connects to its parent branch (or the ground, in the case of the main trunk).

There are four constraints in each joint between bones. These values are set by the user in accordance with the tree type being created (*e.g.,* a willow tree is more flexible than an oak), and can be modified on a per-segment basis using specific *L-System* symbols. Figure 3 provides an overview of the constraints, namely:

- A spherical joint designed to control the angle between two adjacent segments. This prevents movement past a specific limit (constraint 1A in Figure 3).
- A spring force that moves the segments back to their rest positions. The spring is important in giving the branches a sense of flexibility and springiness (constraint 1B in Figure 3).
- A constraint that prevents the segments from twisting around the axis running down the length of the cylinder / capsule, since branches on real trees do not show such rotation (constraint 2 in Figure 3).
- A distance constraint enforcing a minimum and maximum distance between the segments. This is added to prevent strong forces from overcoming the spherical joint's constraints and moving adjacent segments apart, which would create a visible discontinuity (constraint 3 in Figure 3).

The twist and distance constraints can be given a spring value (to better represent branches flexing), but the spherical joint is a simple constraint used to control movement and cannot be given an associated spring value. Each joint has an optional breaking force which, if exceeded, destroys the joint, allowing the tree to break apart.

The breaking forces and all other physics-related parameters are adjustable on a per segment or global basis *via* symbols in the *L-System.*

Current value of all physics parameters are stored, along with the turtle's position and orientation, on the branching stack (*cf.* [Prusinkiewicz and Lindenmayer 1990]), allowing these parameters to be saved and recalled. In addition to symbols setting the values of physical parameters, some symbols specify a factor to multiply a parameter by after each segment has been created. This allows for the creation of realistic tree parameters, for example, applying a factor of 0.9 to the density of a branch, gives that branch a gradual reduction in mass along its length.

For example, the symbol string:

Density(5) DensityRate(0.5) F(10) F(10) F(10)

will create three segments of length 10, with density 5, 2.5, and 1.25 respectively.

While physics engines vary in the features they provide, any reasonably functional engine should support the constraints used here.

The final result of this algorithm is a geometric representation of a tree, as well as a connected underlying physics skeleton that can be used to dynamically simulate and animate the tree. The results of our algorithm can be seen in Figures 4 and 5.

It is also worth noting that our skeleton creation algorithm should operate on any tree generation method in which the trees' structure is explicitly defined. It is not limited solely to *L-Systems* or other descriptive grammars.

3.2 Building Creation: the Shape Grammar Approach

Larive and Gaildrat's [2006] *wall grammar* generates a façade subdivided into many large panels. Each panel represents a portion of the façade's surface, such as a wall, door, window and so on. While this is sufficient for creating a façade, this façade is unsuitable for a dynamics simulation as it would: *a)* break up into large rectangular sections instead of into its constituent material, such as bricks, and *b)* even if the blocks were split further into smaller bricks and other components, they would have no connecting 'mortar' or other internal structure to give the model rigidity and stability. These two concerns are addressed by our extensions, explained below.

Splitting the panels is simply a matter of deciding how to divide them into smaller blocks. We assign types (such as 'wall', 'window', 'door') to each panel. This type contains data such as surface

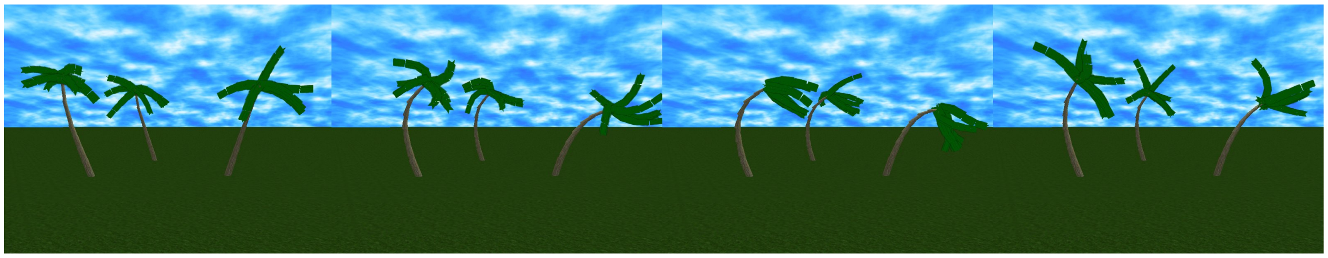


Figure 4: *Four frames taken from an animation of palm trees experiencing a strong gust of wind.*

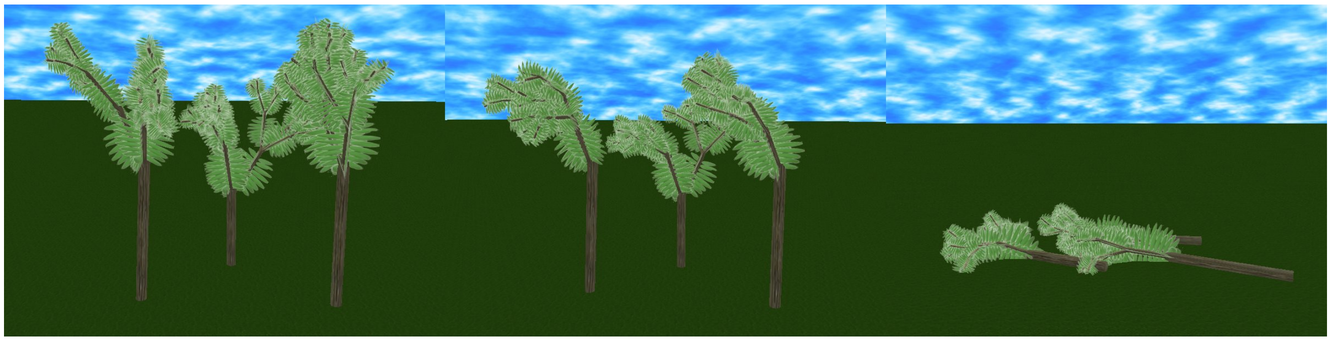


Figure 5: *Three frames taken from a simulation of several trees being knocked over by a super-strength wind.*

texture and material density. How the panels are then split depends on their type (see Figure 6).

For instance, we chose to model buildings made up of brick walls. An important property that needs to be taken into account for brick walls is that bricks are placed in layers, with each layer offset from the layer above and below (see Figure 7). Bricks are joined to adjacent bricks with mortar, which breaks after a certain separating force is applied. That force is linearly proportional to the surface area of the overlapping regions[5]. Walls made of wooden boards can be treated in a similar manner (as ‘longer‘ bricks with different mass). Wood should also fracture and splinter to create realistic breakage animations. While we did not implement this, it should be clear that the wooden boards themselves can be split in much the same way that panels are split into bricks.

One problem with forming panels out of brick is that the bricks may extend outside the boundary of a particular shape / panel and into a neighbouring panel. If the neighbouring panel is of the same type, then this overlap *should* occur, as the two panels should appear continuous. For neighbouring panels of different type, the bricks are cut off at the edges to prevent such an overlap.

Once the panels have been subdivided into their component materials, the material needs to be connected to one another. Without some form of connecting ‘mortar’ holding — for instance — the bricks together, the building would simply collapse.

We do this by determining which of the material components (bricks) are touching each other; these bricks are connected using joints that hold two bricks rigidly together until a strong enough linear or rotational force is applied, at which point the joint breaks.

The linear and rotational constraints are similar to those used in the tree simulation (see Figure 3):

[5]Our building models do not account for long-term stress and fractures, only short-term, violent breakage.

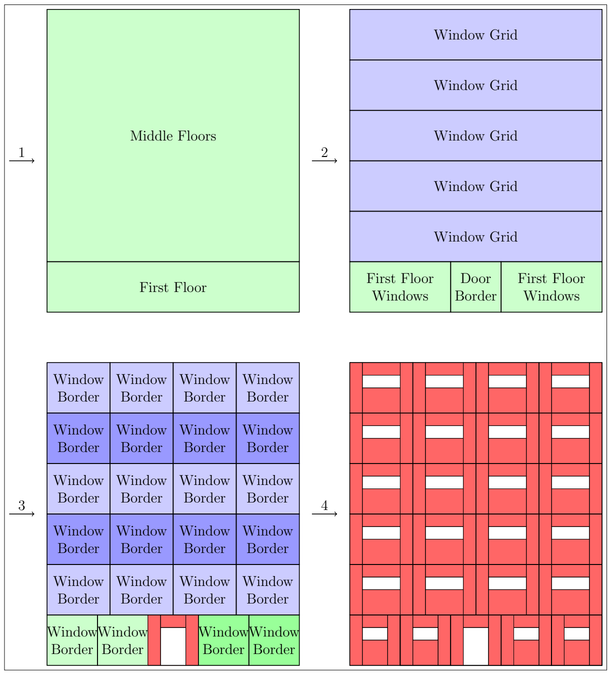


Figure 6: *The façade generation process splits the rectangular area into more meaningful pieces, such as floors. These pieces are split further to create windows, doors and the borders that surround them.*

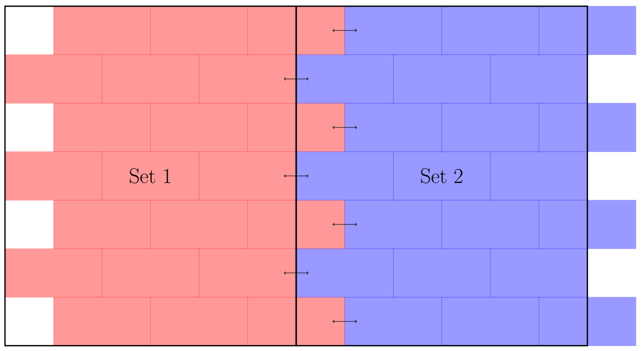


Figure 7: *Both the above panels are tiled with a brick layout, with some overlap. Red block are tiles from the left panel, blue from the right. Since both panels are brick layouts, the brick are permitted to overlap. The bricks on the right edge of the red blocks are connected with the bricks on the left edge of the blue blocks.*

- The joints are kept rigid, allowing no angular variation between the bricks (*cf.* constraint 1A in Figure 3).
- Brick are allowed sufficient force so as to return to their correct positions relative to each other each frame. If this force exceeds a maximum breaking force, the joint is broken (*cf.* constraint 1B in Figure 3).
- A rotational limit stops the bricks from twisting. If the bricks do twist, an opposing rotational force is applied to correct their positions. The joint breaks if it exceeds a maximum breaking force (*cf.* constraint 2 in Figure 3).
- A distance limit constraint attempts to correct any changes in the distance between the bricks by applying an opposing force. If this opposing force exceeds a maximum breaking force, the joint breaks (*cf.* constraint 3 in Figure 3).

In placing joints, the trivial implementation of an $O(N^2)$ algorithm to determine which of the N bricks are adjacent to each other performs poorly for large buildings. Instead, we keep track of which bricks are neighbours when subdividing the panels, thereby avoiding this quadratic cost.

In the special case of bricks lying on the edge of the panel (see Figure 7), we can use information about the panel layout to overlay as necessary the edge bricks of neighbouring panels. Similarly, we can locate the edge bricks of a whole façade during content generation, allowing the edge bricks of neighbouring façades to also overlap.

As the original wall grammar technique does not create any additional structure for its generated buildings beyond a roof, we include an additional step in the process to add a number of internal floors. Most façades are created by splitting a large 2D rectangle into floors. Unfortunately, since the wall grammar only creates façades, there is little data within the generation process that can be used to create further internal structure, such as individual rooms. This means that each level of the building is simply one large room.

3.3 Level of Detail Schemes

We added a number of physics optimisations into our implementation to improve its performance. These consisted of graphics optimisations adapted for use in real-time physics simulation. Additionally, the physics engine we used (Nvidia PhysX[6], but without hardware acceleration, since that was not available on our development OS) provides some optimisations, the major one of which is automatic object 'sleeping' (stopping the simulation of objects that have been stationary for some time until some external force acts upon them).

[6]http://developer.nvidia.com/object/physx.html

Our additional physics optimisation methods include:

- *Frustum culling*: objects lying outside a slightly widened version of the camera's view frustum are not simulated. The frustum is slightly widened to prevent objects spanning the edge of the screen from being affected. This is based on *frustum visibility culling* [Assarsson and Moller 2000].
- *Distance threshold culling*: any object beyond a certain range is not simulated. Because the object is some distance away, the viewer does not notice the lack of simulation. This is equivalent to *billboarding* [Behrendt et al. 2005] in a graphics context.

Both of these optimisations can cause obvious inconsistencies with what the user expects (*e.g.,* stationary distant objects, falling entities not moving when unobserved, *etc.*), so their usage should be restricted to situations where these problems are unlikely to be noticed.

It should be noted that the chosen distance thresholds can have a large impact on both performance and visual quality, and therefore need to be set with care. We chose the threshold values heuristically during development, by examining a range of distances and selecting one which we felt provided an acceptable trade-off between quality and performance.

There remains significant scope for using other level-of-detail schemes, many of which would likely significantly improve performance.

4 Experimental Methods

Our work is intended to answer two main questions, namely, "how realistic and convincing are the results?" and, "How does the system perform, and does it achieve real-time frame rates?"

We examine these questions using three experiments:

- User testing to determine how realistic the animations seem in relation to video of real-world trees and buildings.
- Visual heuristic validation, in which we, the authors, determine how realistic the content is in comparison to reality.
- Performance tests to determine how computationally costly our techniques are.

4.1 User Testing

Our user testing consisted of a study in which we allowed our subjects ($n = 20$; all university students) to compare videos of real-world trees and buildings with videos of our animations.

We recorded videos drawn from across the range of possible simulations our system can produce. These were presented to each user (we use the word interchangeably here with *subject*) alongside videos of similar, real-world events (for example, our system running a simulation of a building being demolished and a real-life video of a building being demolished). The subjects where then asked to rate the realism of our system's simulation in comparison to the real-world video ("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?"). Each subject was also asked to compare

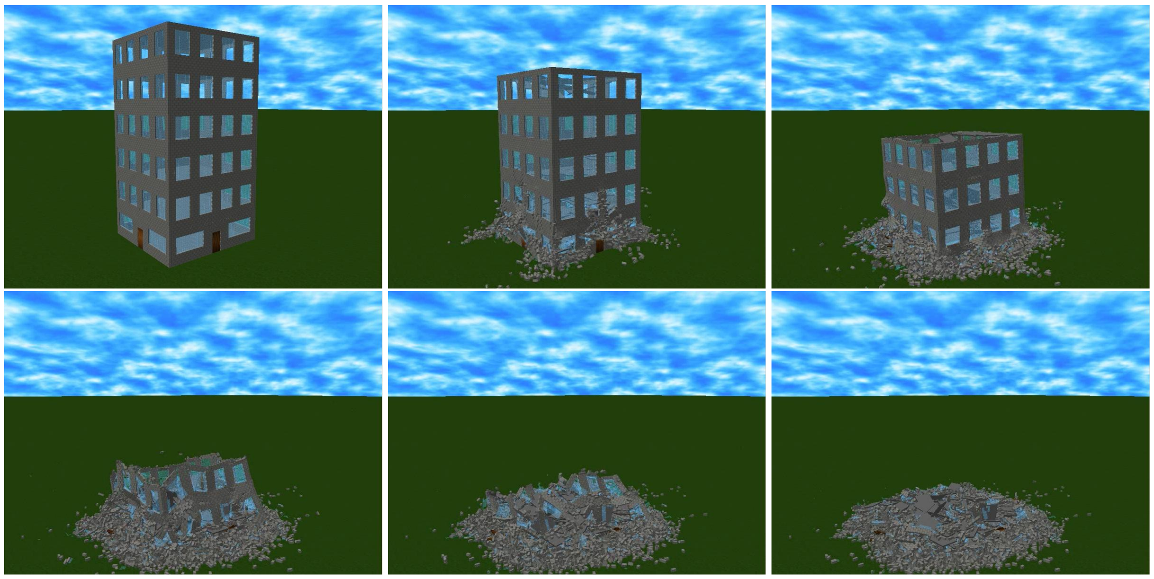


Figure 8: *A building undergoing demolition in the physics simulator.*

the simulation against the dynamics shown in current video games ("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?"). Note that this was not a direct comparison, but rather done from the subject's memory of games they had played. Six of our subjects reported having spent little time playing games in the last few years, and this is taken into account in our results presented below.

The ratings of these videos were carried out using a Likert scale of 1 to 7, with 1 indicating that the simulations were, "Totally unrealistic," and 7 indicating that the simulations were, "Completely realistic." After evaluating the videos the users were finally asked to give their overall impression on the realism of the animations on a scale of 1 to 10 (again, 1 being a complete lack of realism, 10 being full realism), and were also asked whether the animations were an improvement over those seen in current games, again on a scale of 1 ("Current games are more realistic") to 10 ("Massive improvement"), with 5 being a baseline of no change. Each subject was given space to fill in a qualitative assessment of the animations.

The interaction between the test's administrator and the subjects was scripted, and the order in which the videos were shown was randomised. It was emphasised to subjects that they should focus on the movement and dynamics of the trees and buildings, not the visual appearance of the models. 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.

We showed videos of our animations instead of allowing direct interaction with our system for two reasons: firstly, our system could not run large numbers of trees or complex buildings at a frame rate that was acceptable for real-time interaction (see § 4.3 and § 5.3); secondly, it better controls the users' experiences to ensure the validity of our results, removing any variance from the users interaction with the environment.

4.2 Visual Heuristic Validation

In this evaluation we aimed to analyse our generated content with a focus on discovering notably unrealistic elements.

This evaluation was conducted by the authors rather than by test subjects, and was qualitative in manner.

Ideally, we would have carried out an automated analysis based on the laws of physics, but that was not an option due to time constraints. This testing was heuristic in nature, and we searched for elements that stood out as being noticeably realistic or unrealistic.

4.3 Performance Testing

Our implementation, developed in C++, uses OGRE[7] for rendering and PhysX for dynamics simulation. OGRE and PhysX are both software packages that have been used (to greater or lesser extents) in the commercial games industry.

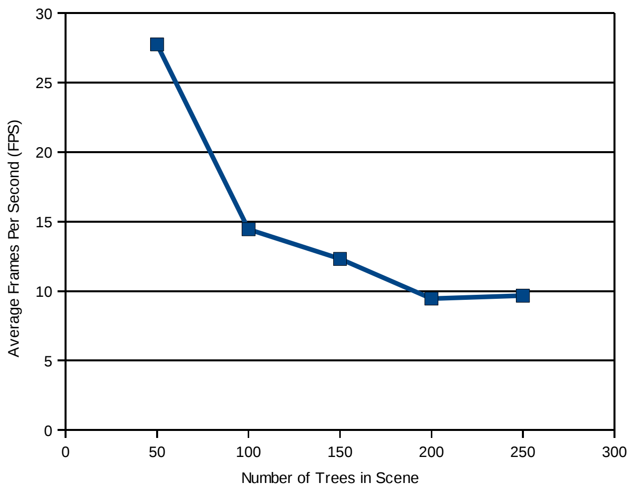
Performance was tested by running the simulation on scenes containing either a single building (and many bricks) or a number of randomly generated trees. Each successive scene increased the number of trees or bricks, in a range of 50 to 250 trees (in steps of 50) and 2000 to 15196 bricks.

Note that the number of polygons per tree varied due to the random nature of stochastic *L-Systems*. On average there are roughly 350 polygons, with a minimum of approximately 100 and a maximum of approximately 750. The number of polygons in a building is always proportional to the number of bricks in the building, there being six rectangles per brick.

Each scene was repeated five times. We recorded and averaged the observed frame rate and the time taken per update of the graphics and physics pipelines. Our software was run on an Intel i7 950 3.03GHz CPU (four hyperthreaded cores) with 6GB RAM and a Nvidia GeForce 295.

Testing was performed with all the previously discussed optimisations and level-of-detail schemes enabled.

[7]http://www.ogre3d.org/



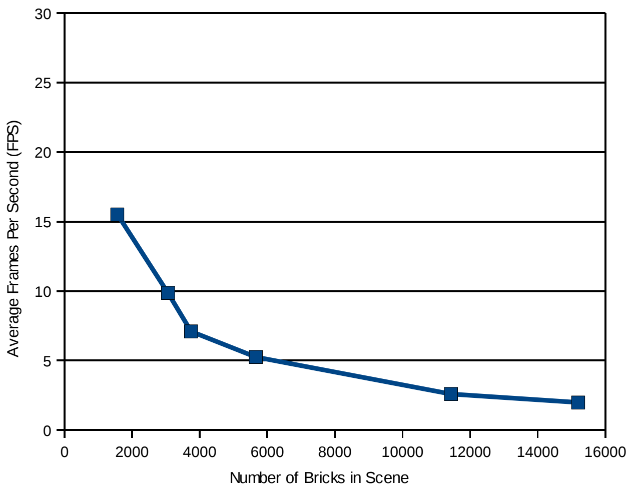


Figure 9: *Graphed results of the performance testing of our system. The y-axis shows the number of frames per second, the x-axis the number of trees / bricks in the scene.*

5 Results and Discussion

5.1 User Testing

All the averaged user scores comparing the different simulations to real-life, except for one, were over 5 (with 5 meaning, “Realistic, but with a noticeable amount of unrealistic elements”). The scores were slightly higher when comparing our simulations to video games, but when removing users who did not often play games there was no significant difference between the results ($p > 0.05$ for each question). Again the average scores (across all users) were, barring one simulation, above 5. When all the data from the individual simulations / video comparisons were grouped, the average score obtained when comparing our animations to real-life videos was 5.26 (standard deviation is $s = 1.34$). The overall average compared to current video games was 5.81 and $s = 1.15$ (a score of 6 means, “Realistic, with some elements that were unrealistic”). There is a significant difference between these composite scores, even when taking into account the users with little recent gaming experience (observed $p = 0.0002$, $t = -3.78$, $df = 267.93$).

These results show that the while there still remain unrealistic elements to our animations, *users considered our animations better than the dynamics of trees and buildings used in video games today*.

These findings were also confirmed by the users’ responses in the overall evaluation questions: no score was lower than 5. The average score compared to reality was 7.6, with $s = 1.19$. For the comparisons to current video games the average was 7.61 and $s = 1.69$. Considering the questions asked to obtain these scores (§ 4.1), it makes little sense to statistically compare them for difference.

On the qualitative side, the test subjects noticed several unrealistic elements in our simulations, but none were considered significant enough to drastically lower the scores. The most commonly described problems with the buildings were that the walls appeared rubbery; there was no dust clouds when buildings were destroyed; and, occasionally, the bricks would float in the air in defiance of gravity. The problems with the trees included violent motions in some of the branches inconsistent with the forces applied to the tree; the leaves of the trees not exhibiting fine-grained behaviour; and the trees not being rigid enough compared to real-world trees.

It should be emphasised that the qualitative feedback included many positive comments, and many complimentary remarks concerning how realistic the trees and buildings were. This, combined with the high scores in the quantitative section, indicate that the negative feedback was not sufficient to detract from the system’s realism.

In general the test subjects were very positive concerning the realism of the models, and felt that our system was a clear improvement over the dynamics seen in current video games. Their feedback did highlight several unrealistic elements, but none of these problems are critical; they are either quirks of the physics engine used in our implementation, or they could be fixed with some tweaking and enhancement of the generative process. For example, trees which are too springy and not sufficiently rigid can easily be remedied by increasing the rigidity of the branch joints (constraint 1B in Figure 3). None of the reported issues were due to problems with the fundamental concepts of the algorithm.

5.2 Visual Heuristic Validation

During this evaluation we identified several unrealistic elements in the behaviour of the generated trees and buildings. Many of these overlapped with the problems reported by the participants in the user testing. However, in our evaluation we noticed a number of problems that the test subjects had not picked up on.

The most significant of these included the larger buildings collapsing on their own due to a lack of internal structure providing support; a slight wobble in the buildings due to a compounded effect of unavoidable, minute bends in the joints (also related to the lack of internal structure); light tree branches occasionally remaining in a state of perpetual motion; and branch segments visibly separating under very high forces.

Most of these problem areas occur only in specific circumstances, and can be fixed by tweaking the procedural generation method in question, or the parameters of the physics simulation. For example, the reported wild thrashing motion of some branches was caused by branch segments overlapping; our solution is to decrease the length of the segments so that this cannot occur.

In general, our content appears realistic, with only a few areas where the behaviour or visual appearance is noticeably unrealistic.

5.3 Performance Testing

We classify performance by frame rate (frames-per-second / FPS) into three categories. They are: real-time (greater than 30 FPS), interactive (10 to 30 FPS), and non-interactive (less than 10 FPS).

As shown in Figure 9, even simple buildings could not be animated in real-time. The smallest case produced a frame rate just above 15 FPS. The remaining test cases all produced non-interactive frame rates. The reason that the number of bricks do not increase in linear steps is due to their number being dependant on the geometry of building, and not directly controllable.

We also did not achieve real-time frame rates for the tree animations. In the smallest case we achieved a result of 27.75 FPS, which is close to 30 FPS. As the number of trees increased, the frame rate dropped off, with the largest two test cases producing results that are just under interactive frame rates.

We do note that our performance testing did not make use of GPU hardware acceleration for the physics simulation. In practice this acceleration can produce a significant speed-up in performance (typically at least 5–10 times). Hence, it is possible that such hardware acceleration would allow our system to produce real-time results for scenes more complex than is currently possible.

6 Conclusion

Procedural content generation is likely to only increase in importance as the need for larger scenes — with greater real-life fidelity — grows. To this end we have shown how to algorithmically produce models that are ready to be animated using physical simulation. Importantly, aspects of the simulation (*e.g.,* specific mass; joint constraints) are easily specified in the grammar itself. Further, a group of users ($n = 20$) found the produced animations to be both realistic — although with some unrealistic elements, most of which appear easy enough to solve — and an improvement on the physics seen in current games.

This work can be readily extended: there exists a wealth of level-of-detail methods that could be implemented to increase frame rates, including using approximate calculations for distant objects, dynamically aggregating collections of objects into a single object for the purposes of simulation, and so on. Performing the physics computation on the GPU would also offer a considerable speedup. Further, the technique of extending the grammars with physical interpretations can be readily applied to procedural models other than that of plants and buildings, such as landscapes (water erosion) and interacting mechanical objects.

Acknowledgements

This work is based upon research supported by the National Research Foundation of South Africa, the Misys Charitable Foundation, the Harry Crossley Research Fellowship, and the David Graaff Foundation.

References

ABELSON, H., AND DISESSA, A. A. 1982. *Turtle Geometry*. MIT Press.

ASSARSSON, U., AND MOLLER, T. 2000. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools 5*, 1, 9–22.

BADLER, N., Ed. 1982. Special Issue on Modeling the Human Body for Animation, CG&A, vol. 2.

BAXTER, R. 2009. Generating hidden structure for procedural scenes. Honours year project, University of Cape Town. Available at: http://pubs.cs.uct.ac.za/archive/00000592/.

BEHRENDT, S., COLDITZ, C., FRANZKE, O., KOPF, J., AND DEUSSEN, O. 2005. Realistic real-time rendering of landscapes using billboard clouds. *Computer Graphics Forum 24*, 3, 507.

BOEING, A., AND BRÄUNL, T. 2007. Evaluation of real-time physics simulation systems. In *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, ACM, New York, NY, USA, 281–288.

BROWN, R., COOPER, L., AND PHAM, B. 2003. Visual attention-based polygon level of detail management. In *GRAPHITE '03: Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, ACM, New York, NY, USA, 55–ff.

CRUMLEY, Z. 2009. Automatic creation of physics components for procedurally generated trees. Honours year project, University of Cape Town. Available at: http://pubs.cs.uct.ac.za/archive/00000590/.

DEUSSEN, O., HANRAHAN, P., LINTERMANN, B., MVECH, R., PHARR, M., AND PRUSINKIEWICZ, P. 1998. Realistic modeling and rendering of plant ecosystems. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 275–286.

EBERT, D. S., MUSGRAVE, F. K., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2003. *Texturing & Modeling: A Procedural Approach*, third ed. Morgan Kaufmann Publishers, San Francisco.

FALOUTSOS, P., PANNE, M. V. D., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 251–260.

GIRARD, M., AND MACIEJEWSKI, A. A. 1985. Computational modeling for the computer animation of legged figures. In *SIGGRAPH 85*, 263–270.

HECKER, C., RAABE, B., ENSLOW, R. W., DEWEESE, J., MAYNARD, J., AND VAN PROOIJEN, K. 2008. Real-time motion retargeting to highly varied user-created morphologies. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, ACM, New York, NY, USA, 1–11.

KALDOR, J. M., JAMES, D. L., AND MARSCHNER, S. 2008. Simulating knitted cloth at the yarn level. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, ACM, New York, NY, USA, 1–9.

KUMAR, S., MANOCHA, D., GARRETT, W., AND LIN, M. 1996. Hierarchical back-face computation. In *Proceedings of the eurographics workshop on Rendering techniques '96*, Springer-Verlag, Porto, Portugal, 235–ff.

LARIVE, M., AND GAILDRAT, V. 2006. Wall grammar for building generation. In *GRAPHITE '06: Proceedings of the 4th international conference on Computer graphics and interactive techniques in Australasia and Southeast Asia*, ACM, New York, NY, USA, 429–437.

LINDENMAYER, A. 1968. Mathematical models for cellular interactions in development, parts I and II. *Journal of theoretical biology 18*, 3 (Mar.), 280–299.

Symbol	Description
Width	Controls branch width.
WidthRate	Each newly added segment's width is equal to its predecessor's width multiplied by this number (it should be less than 1).
Density	Sets the density of branch segments.
DensityRate	Similar to the width rate, except that it scales the density.
SwingLimit	Controls constraint 1A by limiting the maximum swing angle, as seen in Figure 3.
SwingForce	Controls the force of the spring in constraint 1B, (see Figure 3), influencing the springiness of the tree branches.
SwingDamper	Adjusts the dampening force applied to constraint 1B.
BreakForce	Provides a joint's breaking threshold. If the force on the joint exceeds this value, the joint is destroyed (see constraint 3 in Figure 3).
BreakTorque	Provides the breaking threshold for the amount of twist on a joint. If the twist between two segments exceeds this value, the joint between them is destroyed (see constraint 2 in Figure 3).

Table 1: *A table of the of symbols we added to our* L-System *that are used to control the physics simulation parameters of the branch segments. There is scope here for adding additional symbols to control other aspects of the simulation.*

MATYKA, M., AND OLLILA, M. 2003. Pressure model of soft body simulation. *SIGRAD2003, November*, 20–21.

MÜLLER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 154–159.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND VAN GOOL, L. 2006. Procedural modeling of buildings. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, ACM, New York, NY, USA, 614–623.

MVECH, R., AND PRUSINKIEWICZ, P. 1996. Visual models of plants interacting with their environment. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 397–410.

NESME, M., KRY, P. G., JEŘÁBKOVÁ, L., AND FAURE, F. 2009. Preserving topology and elasticity for embedded deformable models. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, ACM, New York, NY, USA, 1–9.

PARISH, Y. I. H., AND MÜLLER, P. 2001. Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 301–308.

POPOVIC, Z. 2000. Controlling physics in realistic character animation. *Commun.ACM 43*, 7, 50–58.

PRUSINKIEWICZ, P., AND LINDENMAYER, A. 1990. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc, New York, NY, USA.

PRUSINKIEWICZ, P., HAMMEL, M. S., AND MJOLSNESS, E. 1993. Animation of plant development. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 351–360.

PRUSINKIEWICZ, P., MNDERMANN, L., KARWOWSKI, R., AND LANE, B. 2001. The use of positional information in the modeling of plants. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 289300.

PRUSINKIEWICZ, P. 1986. Graphical applications of L-Systems. In *Proceedings on Graphics Interface '86/Vision Interface '86*, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 247–253.

PRUSINKIEWICZ, P. 1987. Applications of L-Systems to computer imagery. *Graph Grammars and their Application to Computer Science; Third International Workshop*, 534.

SAKAGUCHI, T., AND OHYA, J. 1999. Modeling and animation of botanical trees for interactive virtual environments. In *VRST '99: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, 139–146.

SELLE, A., LENTINE, M., AND FEDKIW, R. 2008. A mass spring model for hair simulation. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, ACM, New York, NY, USA, 1–11.

SEUGLING, A., AND RÖLIN, M. 2006. *Evaluation of Physics Engines and Implementation of a Physics Module in a 3d-Authoring Tool*. Master's thesis, Umeå University.

STINY, G., AND GIPS, J. 1972. Shape grammars and the generative specification of painting and sculpture. In *The Best Computer Papers of 1971*, O. Petrocelli, Ed. Auerbach, Philadelphia, 125–135.

WANG, H., LIAO, M., ZHANG, Q., YANG, R., AND TURK, G. 2009. Physically guided liquid surface modeling from videos. In *SIGGRAPH '09: ACM SIGGRAPH 2009 papers*, ACM, New York, NY, USA, 1–11.

WATSON, B., MÜLLER, P., VERYOVKA, O., FULLER, A., WONKA, P., AND SEXTON, C. 2008. Procedural urban modeling in practice. *IEEE Comput. Graph. Appl. 28*, 3, 18–26.

WILHELMS, J. 1987. Using dynamic analysis for realistic animation of articulated bodies. *CG&A 7*, 6, 12–27.

WONG, J. C., AND DATTA, A. 2004. Animating real-time realistic movements in small plants. In *GRAPHITE '04: Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, ACM, New York, NY, USA, 182–189.

WONKA, P., WIMMER, M., SILLION, F., AND RIBARSKY, W. 2003. Instant architecture. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, ACM, New York, NY, USA, 669–677.

ZACH, C., MANTLER, S., AND KARNER, K. 2002. Time-critical rendering of discrete and continuous levels of detail. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM, New York, NY, USA, 1–8.