Parallel Lazy Amplification: Real-Time Procedural Modeling and Rendering of Multi-Terabyte Scenes on a Single PC

Carlúcio S. Cordeiro
Luiz Chaimowicz
Universidade Federal de Minas Gerais

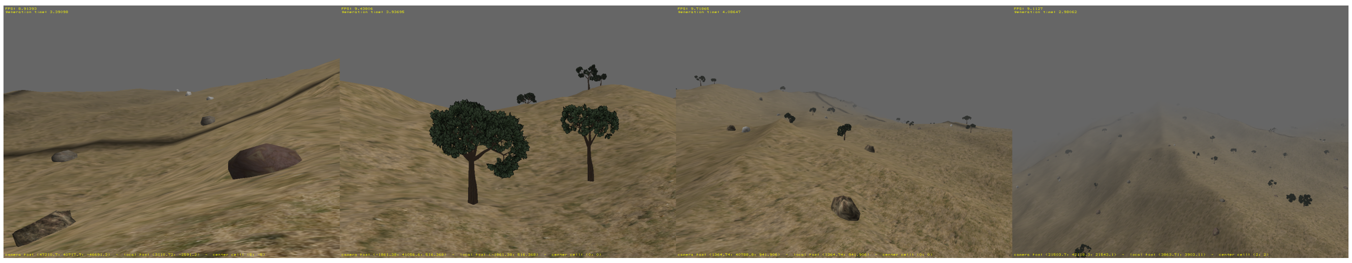


Figure 1: *Some images of our massive procedural world. Although quite simple, the entire scene has about 8 terabytes.*

Abstract

In this paper, we propose a new procedural modeling paradigm called *Parallel Lazy Amplification*. This paradigm may be understood as a combination between two traditional techniques of procedural modeling: *data amplification* and *lazy evaluation*. Data amplification consists in pre-synthesizing the whole geometry before viewing. Alternatively, in the lazy evaluation paradigm, the geometry is generated only when it is needed. The central idea of the paradigm that we propose is to pre-synthesize the geometry that will be potentially seen in the near future and keep it on a cache. A visibility prefetching algorithm determines which models should be generated and which will be discarded. The generation of models is done in parallel with the visualization, without interrupting the system. We implemented a prototype of this paradigm and some initial experiments with a procedural scene of about 8 terabytes showed the feasibility of this new paradigm, especially when performed on multi-core architectures.

Keywords: procedural modeling, geometry management, real-time rendering, parallel computing.

Author's Contact:

carlucio@gmail.com
chaimo@dcc.ufmg.br

1 Introduction

Procedural modeling is an research field in Computer Graphics that includes a number of alternatives to traditional geometric modeling. In this approach, the geometry is specified by a set of parameters and a procedure (algorithm) that creates a model from these descriptions. Some of the main motivations in this area have been the challenge of representing algorithmically the complexity of the objects in the real world both in terms of their form and their behavior. As examples of objects in that class we can mention terrains, vegetation, gases, liquids, fire, architectural buildings, cities and planets [Ebert et al. 2002].

The use of procedural modeling can greatly reduce the modeling time of massive and complex scenes. The use of these techniques is becoming increasingly common mainly in the entertainment industry. For example, procedural modeling techniques have been applied successfully in the movie industry [White 2006].

In computer games, procedural techniques were largely explored in the past. By that time, memory limitations imposed severe restrictions on storage and the use of procedural techniques was a creative way to solve these problems. Some examples includes the game *Elite*, originally published for the former *BBC Micro* in 1984, and *The Sentinel*, a game published for the *Commodore 64* in 1986.

With the modernization and the increase of available memory on computers and video-game consoles, the use of these techniques have been somewhat neglected by the game industry and have not been much used in the top games in the past years. Recently, with the great level of details that the games are presenting today, procedural techniques are becoming more popular again. If before the restriction was the hardware, now the limitation is the growing demand for artists. The relationship of artists by programmer is growing with each new generation of games. Currently, the game studios employ two to three artists per programmer. Thus, procedural modeling help reducing the need for artists generating scenarios automatically.

Another inspiration for this work is the Demoscene [Tasajärvi et al. 2004; Demoscene.info], a digital art subculture that produces audio-visual applications in real time, called *Demos*. This culture has emerged between users of old platforms, such as *Apple II*, *Commodore 64*, *ZX Spectrum* and *Commodore Amiga*. The Demos are applications whose executable code is generally composed of only a few kilobytes. The most common categories are 4 Kb and 64 Kb. They usually do not use any kind of external file, as models, pictures, music and sound. All resources are compressed or synthesized. The Demos are undoubtedly a form of digital art amazing and unique.

Procedural modeling brings at least two major challenges for research in computer graphics. The first challenge is to create procedures and algorithms that synthesize complex and realistic objects and textures. The second challenge is to manage the large amounts of data that are generated by procedural models. This second challenge is relatively less studied and is the main focus of this paper.

In a home PC today, a single procedural model of a terrain, a tree, or a building is easily generated and rendered in real-time. But to generate and visualize a massive procedural model in real-time, as a huge forest, a large urban center, or even an entire planet, more elaborate techniques and tools are required. In these scenarios, the amount of geometry can easily extrapolate the available main memory.

Basically, there are two main techniques for data generation: *data amplification* and *lazy evaluation* (these paradigms will be detailed in the section 3.3). Data amplification consists in pre-synthesizing the whole geometry before viewing while lazy evaluation paradigm generates the models only when they are needed. Lazy evaluation works well for offline rendering. However, to generate all the geometry of each frame in real-time becomes practically impossible. In the other hand, data amplification pre-generates all the geometry and is feasible to be displayed in real-time. But it applies only to models that fit in the main memory.

In this paper, we propose a paradigm that combine data amplification and lazy evaluation. The main idea is to generate the geometry on demand but, differently from lazy evaluation, when the geometry is generated, it will be kept in a cache to be used in subsequent frames. As with data amplification, we have a pre-processing time for building the cache before the beginning of visualization. During the visualization, the system determines which models or parts of it will be seen in the near future and which are already out of context. The models that have low priority of been viewed are discarded, and those who are potentially in the field of view are generated in parallel with the visualization, without interrupting user interaction with the system. We called this new paradigm *Parallel Lazy Amplification*.

The rest of this paper is organized as follows. In Section 2, we discuss some of the main works related to this paper. In Section 3, we do a quick review of basic concepts that are used in the text. The Parallel Lazy Amplification paradigm proposed in this paper is presented in Section 4. Section 5 gives an overview of the graphics engine implemented for the experiments. In section 6, we present and discuss the experimental results obtained. Finally, our conclusions and future work are show in Section 7.

2 Related Work

Procedural techniques have been used throughout the computer graphics history. Many researchers have developed their own procedures to simulate materials, objects and natural phenomena. One of the first procedural techniques used in the graphics community were fractals. Musgrave et al. [1989] describes a model for the synthesis of eroded terrains based on fractals. Later, he described procedural models of a whole planet [Musgrave 1999]. This work was the basis for *MojoWorld* software[MojoWorld] that was originally created by Ken Musgrave.

Another procedural modeling technique is called *L-Systems*, which was originally proposed by Lindenmayer [1968] to model the development of filamentous bodies in a work of theoretical biology. As will be explained later, L-Systems are quite similar to formal grammars used in compilers. The pioneer work in the use of L-Systems in computer graphics is Prusinkiewicz [1986]. The use of this technique is very powerful for the procedural modeling of plants and vegetation, as described in Prusinkiewicz and Lindenmayer [1990] and Deussen et al. [1998]. Inspired by the work of Prusinkiewicz, some authors developed an extension of L-Systems for procedural modeling of cities and buildings called *CGA shape* [Parish and Müller 2001; Müller et al. 2006; Müller et al. 2007]. Like the L-Systems, these techniques are similar to a formal grammar.

Another extension was proposed in [Lluch et al. 2003] to generate procedural models of plants and trees with multi-resolution information embedded in the model. This representation is more appropriate and does not fail in preserving the visual structure of the model, as normally occur with the use of a geometry simplification algorithm.

In an attempt to integrate the characteristics of different procedural modeling systems, Gangster and Klein [2007] presents a new kind of visual language in a single modeling environment. The system shows results of procedural models consisting of buildings, plants and terrains, without the need of external tools.

In the area of representation and management of procedural geometry, Hart [2002] shows how the scene graphs can be used for that purpose. The work presents the paradigms data amplification and lazy evaluation. It also described the Procedural Geometric Instancing technique, which follows the lazy-evaluation paradigm.

Researchers have studied the problem of rendering complex models at interactive frame rates for many years. Clark [1976] proposed many of the techniques for rendering complex models used today, including the use of hierarchical spatial data structures, level-of-detail (LOD) management, hierarchical view-frustum and occlusion culling, and working-set management (geometry caching). Garlick et al. [1990] presented the idea of exploiting multiprocessor graphics workstations to overlap visibility computations with rendering.

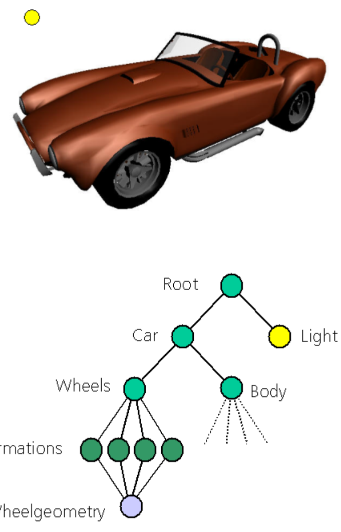


Figure 2: *Example of a scene graph: below is a description of a hierarchical scene and above we have the same scene rendered.*

Correa [2004] introduced a system for interactive viewing of large datasets. The system uses new techniques for out-of-core visualization of models larger than main memory. In a pre-processing phase, a hierarchical decomposition of the model is built using a octree, the coefficients used to test visibility are calculated, and levels of detail are determined. In run time, multiple threads are used to override calculations of visibility, managing cache and rendering.

Out-of-core visualization techniques could be used with data amplification to avoid the size limitations of the main memory. But with this approach the model should be fully generated on disk and pre-processed before viewing. In addition to the generation and pre-processing time, this method would demand extra storage space in disc, which can become a problem for massive multi-terabyte scenes. Pre-generate all the geometry is also less flexible, because the whole model would be completely static. As mentioned, the approach proposed in this paper try to overcome these problems combining Lazy Evaluation and Data Amplification.

3 Basic Concepts

3.1 Scene Graphs

Scene graphs are data structures that are much discussed and researched in computer graphics field. Basically, a scene graph is a spatially consistent data structure, which is used for the representation of three-dimensional virtual environments in computer graphics applications, including procedural modeling. Examples of these structures include bounding volume hierarchies, octrees and grids. Dollner and Hinrichs [2000] present a detailed discussion of scene graphs.

A scene graph is a directed acyclic graph. Each node has a set of attributes that may or may not influence its children nodes. The nodes are organized in a hierarchical fashion corresponding semantically and spatially to the modeling scene.

All scene graph nodes have an attribute called bounding volume. This attribute is a simple volume, usually a box or a sphere, which includes the geometry of all of its children nodes. The bounding volume is used for fast computations of approximate intersection tests of the node with other objects. Intersection tests are mainly used to determine visibility and collision. Figure 2 [OpenSG] illustrates the scene graph concepts.

3.2 Bounding Volumes

The bounding volumes for procedural models can be static or dynamic. A static bounding volume involves all possible geometries synthesized by the procedural model. In other words, a dynamic bouding volume is designed to fit tightly each particular instance of the procedural model. Dynamic bounding volumes are more efficient than static bounding volumes, but they are much more difficult to plan. Dynamic bounding volumes need to be computed at instantiation time.

All procedural models should provide a method for bounding volume estimation from its parameters. Depending on the procedural model, the bounding volume computation can be very simple or very hard. Heuristics to determine the bounding volume should be developed when computing the optimal bounding volume is not feasible.

3.3 Procedural Modeling Paradigms

Virtually all interactive graphics systems follow the same pattern of rendering pipeline, with three conceptual stages: application, geometry, and rasterization. The *user* articulates a conceptual model to the modeler. The *modeler* interprets the articulation and converts it into an intermediate representation suitable for rendering by the *renderer*.

The synthesis of procedural models follows the same data flow. But the way the intermediate representation is generated can follow two different paradigms: *data amplification* and *lazy evaluation* [Hart 2002].

3.3.1 Data Amplification

Models that follow the data amplification paradigm synthesize some sort of intermediary geometric representation. This intermediary representation is generally a description consisting of triangles, polygons or other primitives. Smith [1984] coined the term data amplification to explain how procedural models transform a small amount of data in models with rich details and described by a large amounts of geometry.

The intermediate representation of a complex scene can become very large. For example, a very simple procedural model of a tree, described by only a few floating-point numbers, when evaluated produces a few thousand triangles. A huge forest can easily extrapolates main memory limitations. Data amplification causes data explosion due to the fact that the procedural model is converted into a geometric representation to be rendered. Figure 3 [Hart 2002] illustrates the data amplification paradigm.

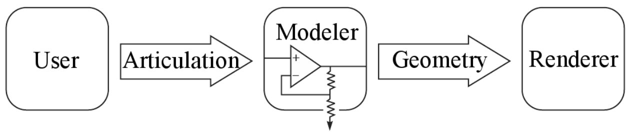


Figure 3: *Diagram that shows the data amplification paradigm.*

3.3.2 Lazy Evaluation

Lazy evaluation avoids the intermediate representation problems that we have with data amplification. The geometry synthesis procedure is performed only when necessary. Procedural models demand a large processing time, then to generate all the geometry each frame in real-time applications is practically impossible. Lazy evaluation keeps a dialogue between the Renderer and the Modeler, as illustrated in Figure 4 [Hart 2002].

Scene graphs also support lazy evaluation for procedural models. For example, the system can generate a bounding volume for a procedural model and perform a test to determine if the geometry contained therein is necessary to render the scene. If the test is negative, the system does not generates the procedural model. The heuristic here is to determine the bounding volume without actually performing the procedure for the model generation.

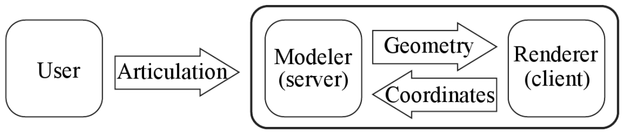


Figure 4: *Diagram that shows the lazy evaluation paradigm.*

3.4 Visibility Prefetching Algorithms

In the proposed paradigm, visibility prefetching algorithms will be used to determine which models, or parts of them should be generated on demand. Visibility prefetching algorithms can be based on the viewpoint of the observer and the bounding volumes [Corrêa et al. 2003]. In out-of-core rendering, the system uses the algorithm to determine the geometry most likely to be seen in the near future, which are read from disk and kept in a cache.

One difference between a out-of-core visualization system and procedural approach, is that in out-of-core visualization we have the whole model a priori. In a pre-processing stage the model is divided in a top-down fashion. In the procedural modeling, the model is generated at runtime. Then the partitioning can be built bottom-up in the model generation.

4 Parallel Lazy Amplification

The main idea of the Parallel Lazy Amplification proposed in this paper is to combine the paradigms of data amplification and lazy evaluation. During the visualization process, the system estimates a set of potentially visible models that will be seen in the near future. Similar to lazy evaluation, the geometry is generated on demand and also in parallel with the visualization. But not only the visible geometry is generated. As with data amplification, parts of the scene are pre-generated and maintained in a cache of geometry to be used in the following frames.

Figure 5 illustrates the Parallel Lazy Amplification paradigm using the same symbolic notation of Hart [2002]. When the *Renderer* needs of a certain model, it makes a request to the *Cache*. To try to ensure that the geometry will be available when requested, the *Cache* uses a visibility prefetching algorithm. When the algorithm determines that a procedural model should be present in the Cache, it makes a request to the *Synthesis Manager*. The *Synthesis Manager* task is to manage and load-balancing the *Modelers*, which are responsible for the generation of models. So far, the prototype implemented uses only one *Modeler* and the *Synthesis Manager* uses a First-In First-Out (FIFO) policy. When more *Modelers* are present, FIFO may not the ideal algorithm because it can generate load unbalance. For example, supose a particular situation when we have 10 models to generate and two modelers. If the fisrt and sixth models takes 0.4 seconds to generate each model, and the other eigth models takes 0.1 seconds each, one good strategy is one modeler generate the fisrt and sixth models and the second modeler generate the other eigth models. In Section 6, we will discuss a strategy for a better distribution of processing between the *Modelers*.

The visibility prefetching algorithm used in our approach is the *Prioritized-Layered Projection* (PLP) [Klosowski and Silva 2000]. PLP is a visibility test algorithm that needs very little processing. The PLP can be understood as the traditional hierarchical frustum culling algorithm, used to discard models outside the field of view. In the frustum culling algorithm, the scene graph is traversed in a depth first order, from the root node to the leaves. If a node is outside the field of view, the node and all its children are discarded. In the PLP, the leaf nodes are kept in a priority queue called front. When a node in the front is visited, it is added to a set of visible nodes. Then, the node is removed from the front and all its neighbors that have not yet been visited are added to the front. PLP requires that each node of the scene graph refers not only to their children but also all to its neighbors.

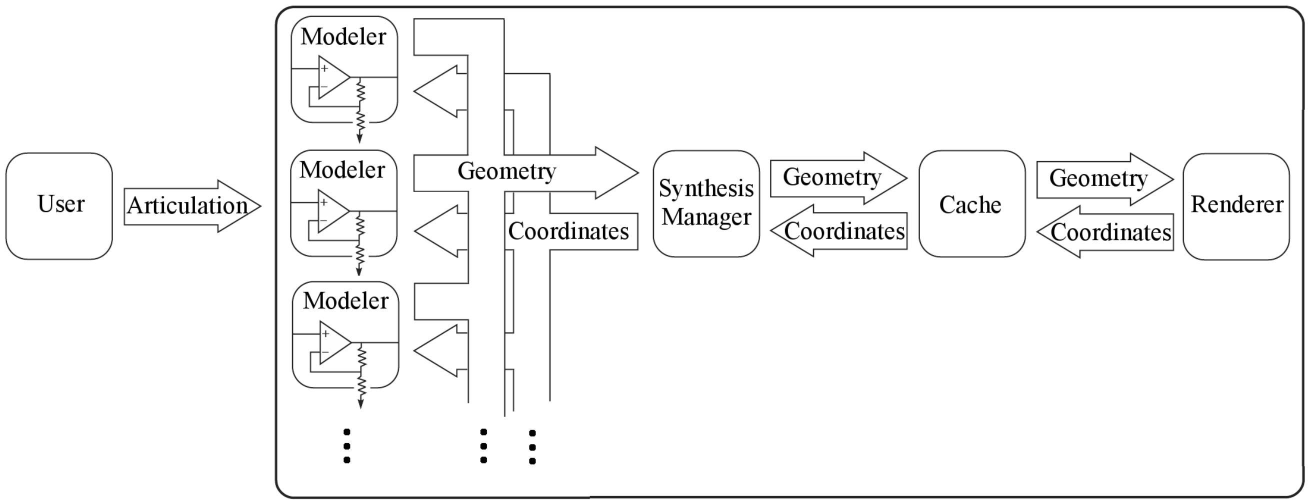


Figure 5: *Diagram that shows the parallel lazy amplification paradigm.*

The key point in the PLP implementation is the heuristic that determines priorities for each node. Klosowski and Silva [2000] presents several heuristics to compute the initial solidity of a node and accumulates it in a certain direction. The solidity of a node is an estimate of how much he occludes an object behind himself. The heuristic that we use was proposed by Correa et al. [2002]. This is a very simple heuristic to determine priorities for nodes. The node containing the camera receives priority -1, its neighbours receive priority -2, the neighbours of neighbours receive priority -3, and so on.

5 Implementation Details

To implement and test the paradigm described in this paper we developed a game engine (PSYGEN) that supports different procedural models. This section presents an overview of this engine and the models that have been implemented. It is important to clarify that it is not the objective of this paper to discuss the architecture of a game engine as the PSYGEN. We will only show some details related to procedural modeling employed by the system.

5.1 PSYGEN

PSYGEN (Procedural Synthesis Graphics Engine) is a graphics engine that allows the implementation and visualization of different procedural modeling algorithms and implements various techniques for real-time rendering. It is composed of several modules that abstracts and facilitates the implementation of procedural modeling algorithms. Figure 6 shows a UML diagram with the organization of the prototype that we implemented.

- **Renderer:** the renderer provides an abstract interface that encapsulates hardware calls and the graphics API. At the present time, PSYGEN has only an implementation of a renderer using OpenGL.
- **Cache:** is responsible for memory managing of the system, maintaining a collection of models and objects potentially seen by the observer. The cache determines which models should be generated, based on PLP algorithm. The cache should also discard geometry with low priority.
- **SynthesisManager:** is responsible for the generation of models. SynthesisManager runs in parallel with the renderer, allowing the generation of models without interrupting the interactive visualization. In the implemented prototype, the Synthesis Manager also plays the role of Modeler. This Manager was implemented using *pthreads*.
- **SceneNode:** the main structure of the system of data visualization of PSYGEN is a scene graph. SceneNode abstract class is the base of all of scene graph nodes. A SceneNode can reference zero or more Shapes, which share the same render state encapsulated by SceneNode.
- **ProceduralModel:** provides an abstract interface to procedural models supported by the system. Some examples of procedural models in PSYGEN implemented so far include trees, terrains and rocks.
- **Shape:** all models and geometry of the system derives from abstract class Shape, for example, the classes TriMesh and Heightfield. These are concrete classes that implement the methods of Shape.

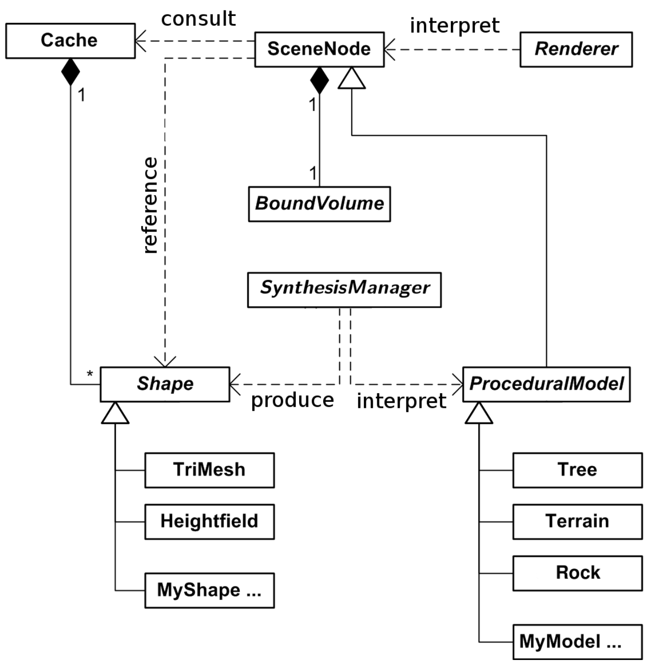


Figure 6: *UML diagram of the main classes involved in the PSYGEN's procedural modeling system.*

5.2 Procedural Models

In this section we presented the three procedural models in PSYGEN implemented so far (rock, terrain and tree) and briefly discuss the asymptotic growth of the algorithms for model generation. Figure 7 shows an image of PSYGEN with the three procedural models implemented.

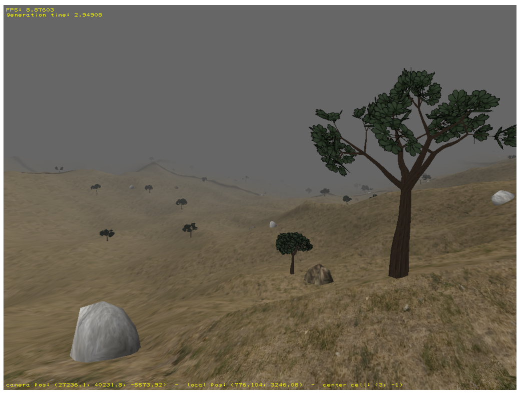


Figure 7: *A view of our simple procedural world. This image shows the three procedural models that we implemented: trees, rocks and terrains.*

5.2.1 Rock

The procedural model of a rock implemented in PSYGEN is a variation of a classic technique know as midpoint displacement or plasma fractal. The difference is that the rock algorithm starts with a tetrahedron and the classic algorithm is done on a plane. Each triangle in the tetrahedron is divided into four new triangles, so that new points are the average points of initial edges of the triangle. After the subdivision, the new corners are disturbed in the direction of the surface normal at the point in question. The parameters of the rock procedural model are:

- **Seed:** Seed to the pseudo-random number generator.
- **Subdivisions:** Number of divisions in which the original mesh will be divided.
- **Radius:** Average radius of the rock.
- **Amplitude:** Disturbance factor in the radius of each iteration.
- **Decay:** Decay factor of the amplitude of each iteration.

The parameter that dominates the generation time is the number of subdivisions. If n is the number of subdivisions and c is a constant that depends on the processor used, the time complexity of rock model is given by:

$$T(n) = c\, n\, 4^{n-1} \tag{1}$$

5.2.2 Terrain

Terrain is probably the most popular procedural model in literature. The PSYGEN terrain procedural model is an implementation of the Ridged Multifractal Algorithm described in Ebert et al. [2002]. The Ridged Multifractal parameters are:

- **Seed:** Seed to the pseudo-random number generator.
- **Heighmap Size:** Size of terrain block.
- **Height:** Determine the largest possible fractal dimension.
- **Lacunarity:** Gap between successive frequencies.
- **Octaves:** Number of frequencies in the multifractal..
- **Offset:** Factor that determines the multifractal characteristic.

The parameters that influence the generation time are the heightmap size and the octaves. If n is the heightmap size and m is the octaves, the time complexity of terrain generation algorithm is given by:

$$T(n, m) = c\, m\, n^2 \tag{2}$$

As the octaves does not change for a particular terrain, we can also describe the time complexity in a simplified form:

$$T(n) = c\, n^2 \tag{3}$$

As the terrain is virtually infinite, it should be built in blocks. We use block sizes of 64 and 9 octaves in our prototype.

5.2.3 Tree

Procedural models of trees are usually implemented using L-Systems [Prusinkiewicz and Lindenmayer 1990]. L-Systems are quite similar to formal grammars used in compilers. From an initial symbol of a particular L-System, the model is derived by a number of iterations. The derivation tree (data structure) represents structurally the model (object tree). The parameters of L-System that we implemented are:

- **Seed:** Seed to the pseudo-random number generator.
- **Iterations:** Number of iterations that will determine the tree height.
- **Branches:** Average number of ramifications. The algorithm select a random value between branches-2 and branches+2.
- **Size:** Size of the first branch.
- **Radius:** Radius of the first branch (trunk).

If n is the number of iterations and m is the average number of branches by iteration, the time complexity of tree generation algorithm is given by:

$$T(n, m) = c\, m^n \tag{4}$$

6 Experimental Results

We performed a set of initial tests and experiments for an preliminary analysis of the proposed paradigm and the impact of the generation of models in parallel with the visualization. To do this we defined a procedural world made of a large terrain divided into blocks. Each block has width 64 vertices, totaling 7,938 triangles per terrain block. For each block, 23 trees were distributed with 3 to 5 iterations and 2 to 4 branches by iteration, and also 40 rocks with 2 subdivisions. Whereas each tree has on average 5,000 triangles and each rock is accurate 64 triangles, each block has about 125,000 triangles. The total size of the world was set to 512x512 blocks. That totals approximately 32,768,000,000 triangles throughout the virtual world. As each triangle has information as its vertices, normal vectors, coordinates of texture and other values (about 256 bytes), the estimated total size of the world procedural hold about 8 terabytes, if it were entirely generated. That means even to limitations of secondary memory (disk) if we deemed current reality of home PCs.

To the first battery of tests, we established a path through which the camera flies in the virtual world. During the camera walk, a log was generated with the measurements of the framerate for each frame along the walk. The experiments were performed on two different computer configurations:

1. AMD Athlon 64 3400+ processor, 1 GB of RAM and a ATI Radeon X800 XT Platinum Edition GPU running Ubuntu Linux 8.04.
2. Intel Core 2 Duo T8300 2.4 GHz, 2 GB of RAM and a NVIDIA GeForce 8600 GT GPU running Mac OS X 10.5.3.

Figure 8(a) shows the results for the first configuration. The result clearly shows that when the generation takes place in parallel with visualization, the framerate drops sharply from about 25 frames per second to less than 15. Though not interrupting the process of interactive visualization, we can clearly see the performance drop.

The same experiment was performed in a dual-core processor (configuration 2). Figure 8(b) shows the results of this second experiment. We can see that when the generation occurs in a multicore processor, the framerate does not drop as in a single core processor. To confirm this assertion, we carried out a third experiment with a configuration similar to the second test. However, this time one of processor cores was off and the test was conducted with only one core asset. The results are shown in Figure 8(c).

The results of this first battery of tests confirms our hypothesis that when the parallel generation is done in multicore architectures, we do not have major impacts on the framerate of the system.

A second battery of tests was performed to confirm the complexity analysis presented at the Section 5.2. For this we measured the generation time of each model for a given set of parameters. These data were used to carry out a curve fitting with their respective complexity functions of each model. The experiments were performed using the first computer configuration. Figure 9 shows the results.

Analyzing the results, we observed that the time complexity of generation algorithms reflects well the generation time. Thus, the system can calculate the constants for each complexity function in the startup. Once constants are computed, the complexity function provides a reasonable estimate of the generation time for a given procedural model. The generation time is a essetial information to have a better load balancing, as we have shown in Section 4.

7 Conclusions and Future Work

This article proposed a new procedural modeling paradigm called *Parallel Lazy Amplification*. In a comparative analysis of parallel lazy amplification with lazy evaluation and data amplification, we can see that parallel lazy amplification can manage large amounts of data that can not be hold by data amplification. We also noticed that lazy evaluation generates all the procedural models whenever they are seen and has no memory management. In a simple situation where the camera is spinning around its axis would do the same models are generated and discarded in a cycle. As the generation of procedural models may require great processing time, this can cause models not shown correctly, models popping in screen and other problems that makes the use of lazy evaluation in real-time impracticable. we believed that we showed the feasibility of the proposed paradigm for the generation and display of massive procedural models in real-time, testing with a very simple procedural world, but that hold terabytes of data if it were entirely generated. The tests also showed that the paradigm is suitable for multicore architectures. Tests done in a dual-core processor showed a minimal impact on the framerate.

The generation time of each procedural model was also analyzed. We showed that the asymptotic growth of algorithms for procedural models provide a good estimate of the generation time. With this estimative we can do a better schedule of models to be generated by available processors and can get a better load balancing of the system.

In a future work, we planned to use the asymptotic growth to determine a good schedule policy for the Synthesis Manager. Other future work includes an implementation with more than one Modeler running on different threads, enabling the generation of more than one model in parallel. We also plan a parallel distributed memory version of PSYGEN to run on visualization clusters. A multithread version is more suited to the reality of home PCs. Dual-core and quad-core processors are already common today. However, We intended to analyze the efficiency and scalability of the system in more than four CPUs, as well as the impact of parallelism in procedural generation. In this scenario, the distributed memory version will be more appropriate, allowing the execution of the system in a visualization cluster with 32 CPUs, for example. Other features planned to be implemented in the future include:

- **Level-of-Detail (LoD) Management:** As procedural models are multiresolution in nature, the system can generate the model with various levels of detail.
- **Geometry Shaders:** The latest GPU generations has a new type of shader called geometry shader. Unlike the vertex shaders and pixel shaders, geometry shaders can create new vertex during its evaluation. This new feature allows the generation of geometry directly into the GPU, allowing a more compact intermediate representation of the model.
- **Mixing traditional models with procedural models:** We also have plan to develop tools and editors to integrate traditional models along procedural models, providing more control to the user in the desired points in the procedural world.

Acknowledgements

We would like to thank Wagner T. Corrêa, for all comments and suggestions.

References

CLARK, J. H. 1976. Hierarchical geometric models for visible surface algorithms. *Commun. ACM 19*, 10, 547–554.

CORRÊA, W. T., KLOSOWSKI, J. T., AND SILVA, C. T. 2002. Fast and simple occlusion culling. In *Game Programming Gems 3*. Charles River Media, 353–358.

CORRÊA, W. T., KLOSOWSKI, J. T., AND SILVA, C. T. 2003. Visibility-based prefetching for interactive out-of-core rendering. In *Proceedings of PVG 2003 (6th IEEE Symposium on Parallel and Large-Data Visualization and Graphics)*, 1–8.

CORRÊA, W. T. 2004. *New techniques for out-of-core visualization of large datasets*. PhD thesis, Princeton, NJ, USA.

DEMOSCENE.INFO. A webportal providing information on the demoscene. `http://www.demoscene.info/`.

DEUSSEN, O., HANRAHAN, P., LINTERMANN, B., MĚCH, 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.

DÖLLNER, J., AND HINRICHS, K. H. 2000. A generalized scene graph. In *Proceedings of Vision, Modeling and Visualization 2000*, IOS Press, Amsterdam, H. N. H.-P. S. B. Girod, G. Greiner, Ed., 247–254.

EBERT, D. S., MUSGRAVE, K. F., PEACHEY, D., PERLIN, K., AND WORLEY, S. 2002. *Texturing & Modeling: A Procedural Approach*, third ed. Morgan Kaufmann, December.

GANSTER, B., AND KLEIN, R. 2007. An integrated framework for procedural modeling. In *Spring Conference on Computer Graphics 2007 (SCCG 2007)*, Comenius University, Bratislava, M. Sbert, Ed., 150–157.

GARLICK, B., BAUM, D., AND WINGET, J. 1990. Interactive viewing of large geometric databases using multiprocessor graphics workstations. In *Siggraph Course: Parallel Algorithms and Architectures for 3D Image Generation*.

HART, J. C. 2002. Procedural synthesis of geometry. In *Texturing & Modeling: A Procedural Approach*, third ed. Morgan Kaufmann.

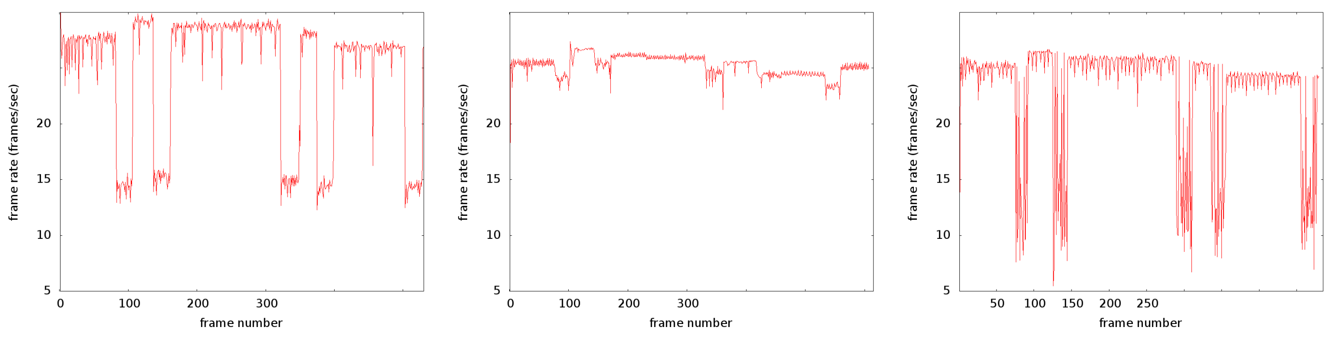
KLOSOWSKI, J. T., AND SILVA, C. T. 2000. The prioritized-layered projection algorithm for visible set estimation. *IEEE Transactions on Visualization and Computer Graphics 6*, 2, 108–123.

LINDENMAYER, A. 1968. Mathematical models for cellular interaction in development, parts i and ii. *Journal of Theoretical Biology 18*, 3.

LLUCH, J., CAMAHORT, E., AND VIVÓ, R. 2003. Procedural multiresolution for plant and tree rendering. In *AFRIGRAPH '03: Proceedings of the 2nd international conference on Computer graphics, virtual Reality, visualisation and interaction in Africa*, ACM, New York, NY, USA, 31–38.

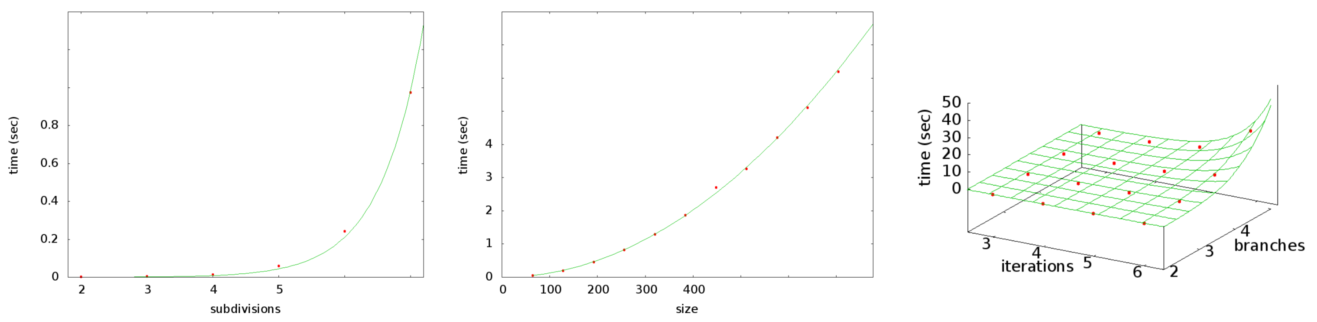
MOJOWORLD. Mojoworld 3. http://www.pandromeda.com/.

MÜLLER, P., WONKA, P., HAEGLER, S., ULMER, A., AND GOOL, L. V. 2006. Procedural modeling of buildings. 614–623.



(a) AMD Athlon 64. (b) Intel Core 2 Duo with two active cores. (c) Intel Core 2 Duo with only one active core.

Figure 8: *Using a multi-core processor to improve frame rates. We measured the frame rates during a 27 seconds walkthrough of the procedural world under three configurations: (a) using a single-core CPU; (b) using a dual-core CPU; (c) using the same dual-core machine used in (b), but with one of the cores off.*



(a) Rock (b) Terrain (c) Tree

Figure 9: *Asymptotic growth for the three procedural models analysed.*

MÜLLER, P., ZENG, G., WONKA, P., AND GOOL, L. V. 2007. Image-based procedural modeling of facades.

MUSGRAVE, F. K., KOLB, C. E., AND MACE, R. S. 1989. The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 41–50.

MUSGRAVE, K. 1999. Building worlds in cyberspace. In *CGI '99: Proceedings of the International Conference on Computer Graphics*, IEEE Computer Society, Washington, DC, USA, 164.

OPENSG. Open scene graph. http://www.opensg.org/.

PARISH, Y. I. H., AND MÜLLER, P. 2001. Procedural modeling of cities. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press, New York, NY, USA, E. Fiume, Ed., 301–308.

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

PRUSINKIEWICZ, P. 1986. Applications of l-systems to computer imagery. In *Graph-Grammars and Their Application to Computer Science*, Springer, H. Ehrig, M. Nagl, G. Rozenberg, and A. Rosenfeld, Eds., vol. 291 of *Lecture Notes in Computer Science*, 534–548.

SMITH, A. R. 1984. Plants, fractals, and formal languages. *SIGGRAPH Comput. Graph. 18*, 3, 1–10.

TASAJÄRVI, L., STAMNES, B., AND SCHUSTIN, M. 2004. *Demoscene: the Art of Real-Time*. Even Lake Studios.

WHITE, C. 2006. King kong: the building of 1933 new york city. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, ACM, New York, NY, USA, 96.