A Hybrid Geometry and Billboard-Based Model for Trees

Marcelo de Gomensoro Malheiros
UNIVATES

Marcelo Walter
UFRGS

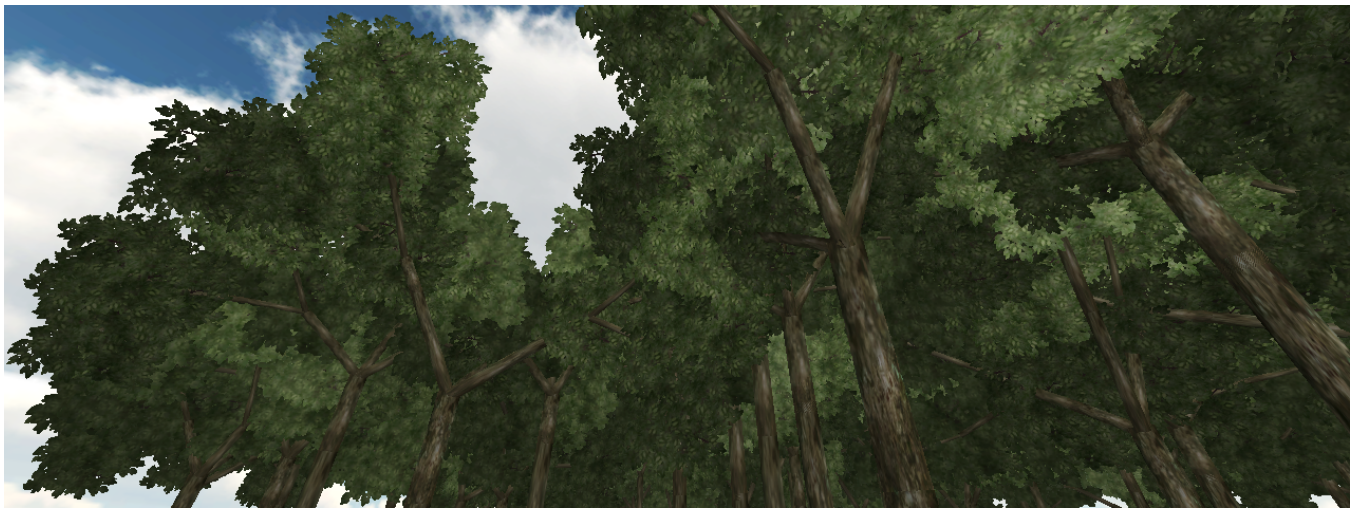

Figure 1: *Trees generated using our technique.*

Abstract

Many applications, such as games, demand massive collections of trees and vegetation in general. The solutions for rendering these environments fall into two possible categories: controlling the amount of geometry displayed, and billboard-based solutions. Using the full geometry is prohibitively expensive when dense forests are needed, therefore level-of-detail solutions are commonly used to reduce the amount of polygons in the scene, however demanding more processing or increased texture memory usage. On the other hand, dynamic billboards rely on render-to-texture operations, which are also costly. We describe an alternative hybrid solution that does not use full geometry, but instead builds trees from textured billboards and simple geometry. By using a simple rule-based L-system and a small set of textures, we can achieve realistically looking visual results of dense vegetation adequate for close to far distance viewing. Our approach provides controlled tree variability and employs fewer polygons than current techniques.

Keywords: Tree Modeling, L-systems, Billboards, Natural Phenomena

Author's contact:
mgm@univates.br
marcelo.walter@inf.ufrgs.br

1 Introduction

Among the many natural phenomena addressed in Computer Graphics, vegetation in general, and trees in particular, have always enjoyed intense research, going as far back as 1984 [Aono and Kunii 1984]. The study of plants might involve the biological simulation of its growth and variability among individuals, the visualization of its structure and the analysis of its morphology. Therefore, it is a multidisciplinary area which also builds from the fields of Biology, Mathematics, Physics and Artificial Life.

There are several applications which demand procedural creation of vegetation, but probably the most common is to populate virtual worlds. Flight simulators, scientific visualization, digital agriculture, and game systems that need to create and simulate a virtual world, usually have plants as one of its building blocks. Such environments often use not single trees, but massive collections of trees instead, in order to render realistically looking forests. The research in graphics on this topic progressed from modeling and rendering individual trees [Aono and Kunii 1984], to real-time dense coverings of a terrain with trees and plants [Gumbau et al. 2010; Deng et al. 2010].

Today, most of the visualization and gaming applications run in real-time, taking into account the advances of GPU processing and the increasing computational power of multi-core CPUs. Therefore not only the visual representation of a plant or tree needs to be adequate, it should also be integrated within a complex virtual world that must be rendered on demand. Therefore, the goal for this work is to simplify tree generation, focusing on the 3D visual result and not in the geometric accuracy. We thus want to achieve an adequate visual representation of trees with a much simpler model, combining the generative power of L-systems together with the smart use of textures. We argue that our solution provides similar perceptual richness as the standard full geometric model, but with lower computational costs. In Figure 1 we illustrate a forest rendered with our method, whereas the geometry for one tree is shown in Figure 2.

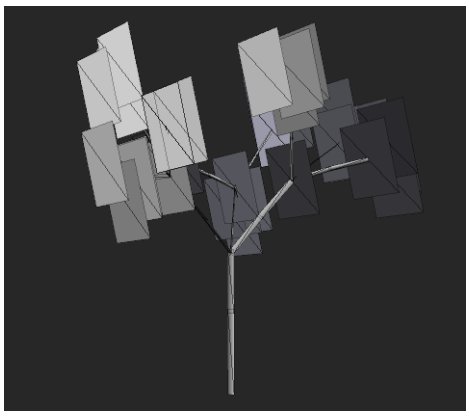

Figure 2: *Simplified geometry for a single 3D tree.*

The rest of this paper is organized as follows. In the next section we discuss the related work, detailing some limitations of current techniques. A more in deep presentation of L-systems is given in Section 3. After, in Section 4 we present our approach using textures as building blocks. In Section 5 we do a brief visual comparison against a model generated by a current professional game engine. Finally, in Section 6 we conclude by summarizing our contributions and discussing further research directions.

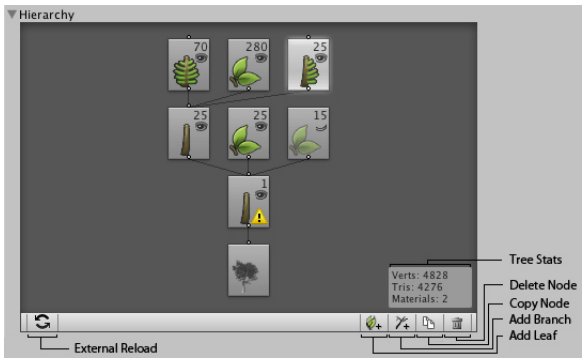

Figure 3: *Tree Creator tool from Unity game engine.*

2 Related Work

Individual trees There is a very extensive literature addressing the many aspects of modeling, rendering, and animation of individual and collection of trees, going back as far as 1984 [Aono and Kunii 1984]. Early approaches applied a diversity of techniques to get the visual representation of vegetation. In [Reeves and Blau 1985] the first simple particle system was used to model trees and grass. Later, [Oppenheimer 1986] derived a parametric self-similar model based on the notion of a fractal structure. [Weber and Penn 1995] proposed a generic parametric geometric model for branching plants, that could account for a large variety of species. This model employed specific controls for the main trunk, for each one of the several levels of branches, and for the leaf placement. It was one of the first models to take into account the need of simplified rendering for far-viewing trees. This is particularly interesting, because all the levels are derived from the same geometric description, which can then vary smoothly in the visual sense from the simplest to the fullest representation. However, the models created are very complex, a complete tree ranging from 5000 to $100,000$ polygons. Although not explicitly stated, current specialized modeling tools such as Xfrog or the tree generator module from the Unity game engine (shown in Figure 3) seem to be heavily inspired by [Weber and Penn 1995], as the different hierarchical components are modeled separately, and the hierarchy itself can be deepened to include more detail. L-systems [Prusinkiewicz et al. 1988] are also heavily used for modeling and rendering a variety of phenomena from the Plant Kingdom. They express the main morphological structure of various organisms using formal grammars. Even though it started as a simple model, it has been extended over the years to cover much more complex models [Prusinkiewicz et al. 1996] and simulate diverse botanic characteristics [Federl et al. 2003].

Collections of trees Other works focused in different approaches for the visualization of large masses of vegetation, taking into account occlusion effects in dense arrangements and adaptation to the terrain topography. [Gardner 1984] employed procedurally textured polygons to account for color variation on the top of trees. Later research used precomputed views of trees [Max and Ohsaki 1995] and generalized image caching [Shade et al. 1996] to speed-up the navigation of virtual scenes, combining both geometry and textured polygons. Replacing geometry by images is a very attractive idea, usually known as *billboards* – flat images always oriented to face the camera – or *impostors* (images which do not change orientation). This is a very simple and low-overhead technique, quite common as solution for displaying trees and vegetation in video games. The seminal work on impostors was presented by [Sillion et al. 1997]. The main idea is to replace geometry seen afar by images of the same. Many extensions and improvements were later presented by [Day and Willmott 2005]. The concept of a structured collection of billboards is known as a *billboard cloud* [Décoret et al. 2003]. [Behrendt et al. 2005] applied the idea of billboard clouds to render trees. This work was later extended to provide level-of-detail by [Lacewell et al. 2006].

Few approaches addressed the specific goal of real-time rendering large collections of trees and vegetation. Both [Meyer et al. 2001] and [Qin et al. 2003] addressed the specific problem of illumination of such complex environments. [Szijarto and Koloszar 2004] presented a 2.5D billboard-based approach, similar to the one used in this paper, whereas [Lluch et al. 2004] used L-systems and impostors for accelerated foliage rendering. An interesting solution using points as primitives was presented by [Gilet et al. 2005], and more recently the work by [Livny et al. 2011] introduced faithful reconstruction of trees from collection of points, where the complicated dense geometry is abstracted by geometric structures called lobes.

More recently, GPU-based approaches were presented for tackling the complexity of rendering in real-time dense coverings of vegetation. [Deng et al. 2010] introduced a purely geometry-based approach, where a GPU-specific data structure is introduced to allow for faster renderings and less CPU-GPU communication. [Gumbau et al. 2010] explored massively parallel architectures to improve efficiency when displaying forests using an adaptive level-of-detail approach based on camera position.

The idea of billboarding, together with a simplified geometry model and the use of appropriate textures to account for visual details, is the basis of our work, detailed in Section 4.

2.1 Discussion

For real-time visualization, both for scientific purposes and for gaming, the rendering of large collections of trees is normally split into two situations. First, a full-detail geometric tree model is used for close-up views. Such models typically employ a large number of polygons and are UV textured with high quality images. The other case happens for distant trees, which are rendered as billboards, either by using predefined textures or by rendering the full model into a new texture, which is then mapped into a quadrilateral.

The use of predefined textures should be used carefully, as they should visually match the tree geometric model. Therefore a render-to-texture preprocessing can be a better alternative, which can give a more faithful representation for billboards. However, two main problems can arise from this combined approach:

- First, there should be a transition somewhere in the ray pointing away from the observer to tree, where the geometric model is replaced by the flat billboard image. If the change happens too close to the viewer, the billboard illusion is broken. If it is done too far, some processing power may be wasted for rendering the geometry, as the tree now occupies only a small space on the screen. Although a geometric level-of-detail control can reduce the second situation, it also adds another layer of complexity to the scene rendering, as the geometry should continually change for each tree displayed.

- Second, and most significant, the usual approach lacks the individual variation of natural trees. Even within the same species, many different morphological and visual details vary greatly among distinct trees. Having just one geometric model can be too limiting when dealing with a whole forest filled by such species. And the similarity may be more accentuated for far away trees, as the billboards will look too similar and a repeating pattern may emerge.

Actual rendering systems deal differently with these problems. We specifically analyzed the workings of the Unity game engine [Unity 2011] in relation to handling tree modeling and rendering. This engine employs the model to billboard transition described previously, which is visible on common scenarios.

Although Unity includes the Tree Creator tool, any geometric tree model can be used in games. In fact, the game engine actually uses only static meshes, so any controllability that is given by the Tree Creator is only useful during the authoring process. Once a tree is inserted into a scene, it cannot be altered, except for very simple variations, discussed below. So if some variations from a given tree species are needed, several meshes need to be generated and loaded in the game. Unity adds some diversity when placing large quantities of trees over a terrain by controlling the amount of color variation, tree height, and tree width. However, even with such global changes to the model (or billboard), several characteristics like overall shape or orientation do not change.

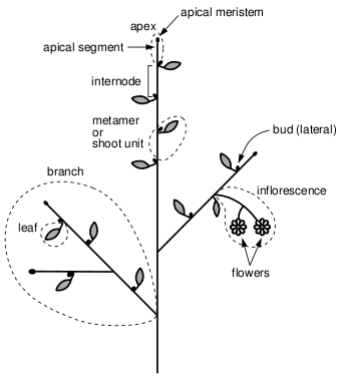

Figure 4: *Modules describing plants [Prusinkiewicz et al. 1996].*

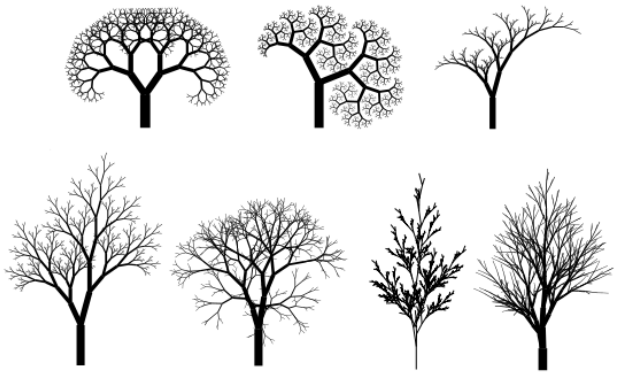

Figure 5: *Some 2D L-systems [Prusinkiewicz et al. 1996].*

In this paper we address the problem of tree generation from close to far distance, using an alternate model that accounts for visual variability and low rendering overhead.

3 L-systems

L-systems describe a broad range of methods that model plants based on formal grammars. The concept of using grammars for the morphology and development of organisms was introduced in [Lindenmayer 1968]. Initially it was used to model multicelular organisms that form linear or branching filaments. The organism is itself treated as an assembly of discrete units, called *modules*. Such formalism does not imply any nature for the modules. Therefore, each module can represent a single cell in simple organisms, or complete functional units in more evolved species. For example, for higher plants each module can represent apical meristems, internodes, leaves and flowers [Federl et al. 2003].

In the seminal paper for L-systems in Computer Graphics, [Prusinkiewicz et al. 1988] describes a simple rewriting scheme, composed of a set of symbols and a set of rules that are iterated over an initial symbol. Each symbol is actually a module, and the iterative process mimics the development of an organism. For example, some typical morphologic elements represented as modules are shown in Figure 4.

The idea of a rewriting system is to continually replace a single module for a set of modules, in parallel. That is, the substitution occurs simultaneously for each symbol at a given step. This simple procedure, more strict than the traditional rules from formal grammars (where just one substitution can happen at a time), actually mimics the growth of the plant and both preserves the overall structure and adds elements of self-similarity. Such earlier L-systems are dubbed *DOL-systems*, for being both deterministic and having 0 interactions. That is, exactly one rule applies to any symbol. Each rule is context-free and assumes the following form:

$$predecessor \rightarrow successor$$

After running some iterations, the resulting set of symbols can be interpreted. One particular interpretation adds "turtle graphics" meanings to the symbols, with letter F meaning to go forward and a $+$ sign meaning to turn left by some fixed angle. For branching structures, the symbols [and] are added, meaning pushing into and popping off the current drawing state from a stack, respectively. Thus it is possible to derive an independent branch somewhere in the structure, and later return to its branching point to create another one. Such variation is sometimes named a *bracketed L-system*. Although very simple in nature, such systems can generate a large variety of branching structures, both artificial-looking and similar to natural plants. Some examples are shown in Figure 5.

Over the years, many variations were proposed to enhance the expressibility of L-systems or adapt them to specialized uses. One common generalization is called *parametric L-systems*, where parameters and function evaluation are added as part of the rule rewriting steps. In this scheme, a rule is now expressed as:

$$predecessor : condition \rightarrow successor$$

Here the condition is first evaluated and then such rule is only applied if the test is true. Such system is therefore more complex, but gives more control of specific features on the resulting representation. Note, however, that this extension still makes the grammar context-free. Thus with few additions the system could be further improved by adding context-dependent evaluation [Prusinkiewicz et al. 1996].

Of note, L-systems have been recently used for broader applications, unrelated to the botanical field. For example, in [Parish and Müller 2001] L-systems were used to generate city maps and follow terrain constraints. [Prusinkiewicz et al. 2002] describes the specification and generation of subdivision curves using parametric context-sensitive L-systems with affine geometry interpretation. And the inverse process, of designing a plant overall appearance and then deriving the corresponding set of rules was described in [Anastacio et al. 2008].

For our work, which focus on simplicity to reduce real-time load in modeling and rendering, we used a bracketed L-system with just a few parameters. A more complex rule system could be useful to get more fine-grained control in the future.

4 Our Approach

A key insight for our proposal is that the visual result for a tree can be achieved through smart use of texture overlaying. Therefore, we describe two variations of our technique, the first one being more suitable to far range viewing, providing a simple method to generate varied tree billboards using simple 2D drawing operations. For close and medium distances a simple 3D model can be created, still resorting to the use of overlaid textures for low processing cost.

Either way, we believe that the smart use of observer-aligned textures can be visually enough to characterize trees, provided a simple underlying geometric description based on L-systems is available.

4.1 Tree model

We model the tree structure by a simple bracketed L-system, which comprises a set of symbols and a set of production rules. The model needs to be evaluated only once for a given species, as the resulting string of symbols represents the morphologic modules of a specific "canonical" tree.

We explicitly choose to keep the L-system deterministic as the individual variability can be introduced later, just before rendering, thus producing controlled variations of the basic form.

For example, a set of rules like

Figure 6: *Turtle interpretation of a 2D L-system: set of line segments (left) and rendered using only two distinct textures (right).*

Figure 7: *A 2D rendered plant with enhanced textures.*

$$X \rightarrow F-[X+X]+F[+FYX]$$
$$F \rightarrow FF$$

would result in the following model, given X as the start symbol and performing two iterations:

$$FF-[F-[X+X]+F[+FYX]+F-[X+X]+F[+FYX]]$$
$$+FF[+FFYF-[X+X]+F[+FYX]]$$

The meaning is that F stands for a trunk or branch segment, Y for a leaf, $+$ and $-$ for relative rotations, and $[$ and $]$ for state push/pop actions. Note that X is just a non-terminal symbol, and thus does not represent any tree feature. If we remove X together with bracket pairs that do not have features inside and other redundant groupings, we have:

$$FF-[FF[+FY]+FF[+FY]]+FF[+FFYFF[+FY]]$$

This representation can be interpreted as standard turtle instructions. For example, a further expansion of this set of rules will generate the set of segments shown in the left of Figure 6.

The use of $+$ and $-$ rotations yield a 2D model, which may be enough for trees being visualised far away. For a 3D model, we simply augment the basic turtle commands, adding & (pitch down), ^ (pitch up), \ (roll left) and / (roll right), all relative to the 3D "forward" direction of the turtle [Prusinkiewicz et al. 1996].

4.2 Far range viewing

Traditionally, trees being viewed far away are shown as billboards, as the amount of detail needed is low and the used screen space is small. However, either the billboards are made of predefined textures or they are generated on demand from complete geometry trees (by render-to-texture operations). Predefined textures lack flexibility and visual variation when a large set of trees is needed, while on-demand rendering can be too costly for such small scene elements.

We should note that in a far range viewing scenario, the fine details of a tree are lost in a compact and dense cluster of foliage. Also, the inner structure of thinner branches is no longer visible due to its size or occlusion by the leaves. At this range, the parallax effect between tree elements is no longer significant. Nevertheless, the overall tree structure usually needs to be consistent with the closer view. At this range the leaves will be reduced either to small dots or blended into a varying mass of foliage. So in this level many elements can be thrown out or simplified, but the overall morphological structure and visual result must be kept.

We propose that a simple 2D rendering can be used for this case, using the result from the L-system expansion described earlier. The drawing can be simplified by the use of separate textures for the trunk and branches and for the leaves. Careful choice of textures can provide most of the visual detail associated with a real tree, however requiring significantly fewer resources. Since all geometric information can be derived on demand as the generated tree model is parsed, there is very little extra storage needed for the positioning or size of the tree elements. Therefore, such elements can be rendered by simply placing textures aligned and resized to match the modules created.

As an added detail, leaves should be represented by adequate transparent textures, whereas the branches should use a rectangular texture correctly aligned. The right side of Figure 6 shows a very simple rendered plant model, where only two textures were used. Note that there is no explicit junction between trunk segments, where by gradually reducing the width of the segments as the branching level increases, most of the joints are visually smooth.

Also, given that we only apply simple operations as scale, rotation and stamping (using alpha) to the textures, these can be easily implemented either in software or accelerated by hardware. This approach can be used both to cheaply compute billboards or draw on demand directly to the screen. For 3D setups, the only requirement is that both the billboards and the textured polygons be facing the camera. The particular strategy depends on the implementation of the visualization system.

In order to enhance the visual results, we generated a second set of textures. We applied a simple shading to the trunk texture using a horizontal darkening gradient and darkened the leaf texture by 50%. The shaded trunk texture was used as it is, while both leaf textures were selected randomly during rendering. Figure 7 presents an example of this, comprising of 28 segments and 42 leaves.

4.3 Close to medium range viewing

For close viewing, a detailed 3D model of the tree is clearly needed, both to get the adequate parallax effect (as an element of the scene), and to visualize the detailed structure of trunk, branches and leaves. Depending on the application, this near-camera representation can be simplified to reduce computational cost in exchange for less realism. For medium range viewing, however, such simplification is already enough to represent the full tree.

Therefore we can also use an L-system for those situations. We then need to augment the rules to take into account rotation over the three axis. The main difference is that we should retain the 3D positions of the modules of the tree. In particular, we need to store both endpoints of the trunk and branch segments, and the center of the leaves. During rendering, a leaf is drawn facing the camera,

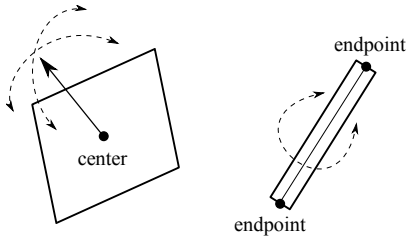

Figure 8: *Degrees of freedom for a leaf (left) and for a trunk segment (right), when facing the observer.*

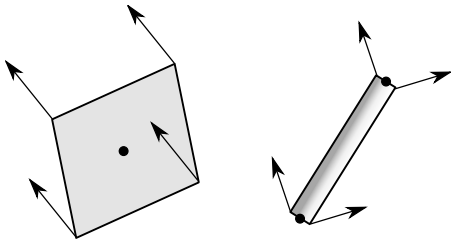

Figure 11: *Additional shading given by definition of normals: precomputed for a leaf (left) and dynamic for a trunk segment (right).*

rotating freely but always centered on the stored point. A segment, however, should follow the same endpoints, and swivel around its axis to face the camera. Both situations are schematically illustrated in Figure 8.

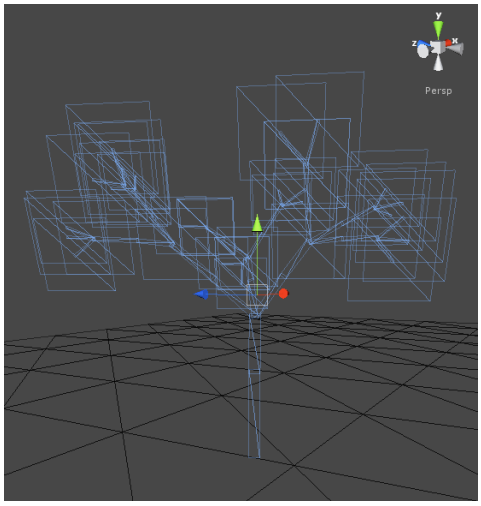

Figure 9: *Wireframe view of a 3D rendered tree.*

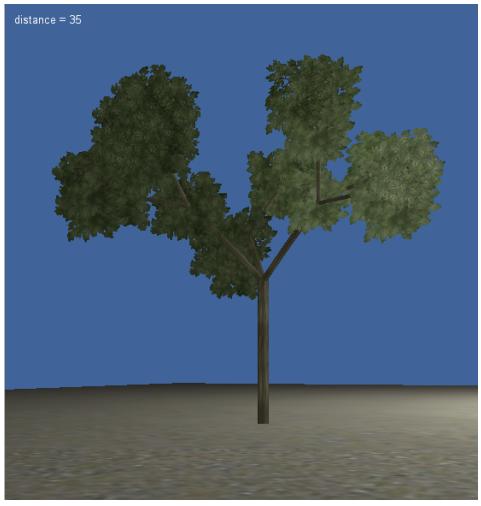

Figure 10: *Textured view of a 3D rendered tree.*

That way, even if the observer changes position, we can have maximum coverage for the leaves by having them always face the camera. By rotating the segments and keeping their endpoints, the tree structure is maintained, even if we use just a set of flat rectangular textures. An example of a resulting 3D tree is shown as wireframe in Figure 9 and fully textured in Figure 10.

Two specific techniques were used to add shading variation to the textures employed in the resulting tree, and to provide volume cues.

For the leaves, the normals were computed at tree creation time, and kept unchanged, even as the corresponding textured quadrilaterals rotate to face the observer. First, after all leaves centers were already placed, we computed their centroid. Then, each leaf normal was assigned to point out from the centroid to the corresponding leaf. That way each leaf texture would account approximately for some solid angle of the sphere that circumscribes the tree, and be shaded accordingly. This provide a consistent shadowing, done by the graphics pipeline, which follows the illumination of the scene, actually simulating the volume of the foliage. We employed just one normal per textured leaf polygon, but the visual result could be enhanced a little by calculating the normal for each quadrilateral vertex instead. This situation is shown by the left of Figure 11.

For the trunk and branch segments, we used a different strategy. To simulate the appearance of a curved surface, the normals at the quadrilateral vertices were updated as the polygon rotated, to always diverge from the camera direction. For example, given a segment facing the observer, the two vertices on the left side would have the same normal, pointing almost to the left. The same applies to the right side, with rightmost pointing normals, as illustrated by the right of Figure 11. The results is that the illumination will be interpolated along the texture surface, leading to a shadowing that suggests that the segment is curved.

In summary, for the 3D case, the leaves need to have only their vertices updated when the camera moves, always turning around their centers. For the trunk and branch segments, both the orientation and normals should be updated.

Note that depending on the viewing distance, the amount of calculation for these updates can be reduced. From a closer view, it is better to have all the tree elements individually changed to face the camera, as discrepancies on the alignment would be noticeable. For a medium range viewing, it is enough to establish just one main direction from the tree to the observer, so that all the leaves and segments can be adjusted by it, thus reducing the computational cost of the update.

It should be observed that this 3D approach is also suited for the far range situation, when there is need for consistency between the visual appearance of the trees in all distances.

4.4 Tree variability

As we discussed earlier, one of the drawbacks of the traditional approaches for tree rendering is the cost of adding variability. This either implies the generation of new complex geometry for new individuals of the same species, or demands the storage of many different textures for billboards.

In our approach, by using the underlying high-level structure given by the L-system, we can achieve variability by just adding random changes when we interpret the resulting sequence of symbols and generate the simplified tree geometry. So the variation is added just before the actual rendering, when normal and vertex data are fed into the graphics pipeline.

We therefore identified two strategies. First, we added random angle turns to segment joint, which accounted for some slight turns and curves on the combined branches. The idea is to mantain the overall tree structure the same, but add variation in the bending of all elements. Of course, the impact will depend on the magnitude of the variation. For our specific experiments, we employed angle variations in the interval $[-10, 10]$.

Figure 12: *Variations of a 3D rendered tree.*

The second strategy was to random skip some symbols from the L-system result, therefore ignoring the corresponding geometric transformations. Both leaves, trunk, and branch segments could be skipped, given some probability of occurrence. This added a higher level of irregularity to the generated trees, which impacted strongly on the appearance of the individuals, effectively changing the underlying tree structure. For the examples in this paper we used a $1/20$ probability of being skipped, and some results are shown in Figure 12.

5 Results

We implemented our technique using the Unity game engine.

As a first evaluation, we ran a visual comparison between our resulting tree and one of the Unity bundled tree models, called Alder. The Unity tree is a geometric model comprised of 1797 vertices and 1044 triangles. Our goal was to achieve a visually similar result, as seen in medium range, and employing the same textures as the stock model. The comparison is shown in Figure 13, with the Unity model displayed both with and without the engine specialized tree shader. The generated tree has 50 segments and 39 leaves, therefore employing only 89 textured polygons overall.

In Figure 14 the close-up of another rendered tree is shown, with the same complexity as the previous one. We used a single 1024x1024 texture comprising a bark pattern and three leaf arrangements, as shown in Figure 15. Finally, in Figure 16 we display a forest with 100 individual trees.

Regarding the performance, only preliminary tests where executed, as the implementation was prototyped using the JScript programming language inside Unity. Because JScript is interpreted, there is a processing overhead on the CPU unrelated to the actual rendering, which resulted in low framerates. We are currently addressing this situation, by moving the geometry updating solely into the GPU.

6 Conclusions

We have presented a new hybrid approach for tree rendering, which uses very few resources, can be hardware accelerated, provides variability without the need to reevaluate the model, and results in adequate visual detail for close to far distance viewing ranges. Usually a 2D evaluation is enough for far distance, but a simplified 3D model is needed when the tree is close to the camera.

As future work, we plan to explore the implementation of this technique in GPUs and on adding other desirable features, such as support for smooth level-of-detail, wind movement, and also simple shadow casting.

References

ANASTACIO, F., PRUSINKIEWICZ, P., AND SOUZA, M. C. 2008. Sketch-based parameterization of l-systems using illustration-inspired construction lines. In *Proceedings of the EUROGRAPHICS Workshop on Sketch-Based Interfaces and Modeling*.

AONO, M., AND KUNII, T. 1984. Botanical tree image generation. *IEEE Comput. Graph. Appl. 4* (May), 10–34.

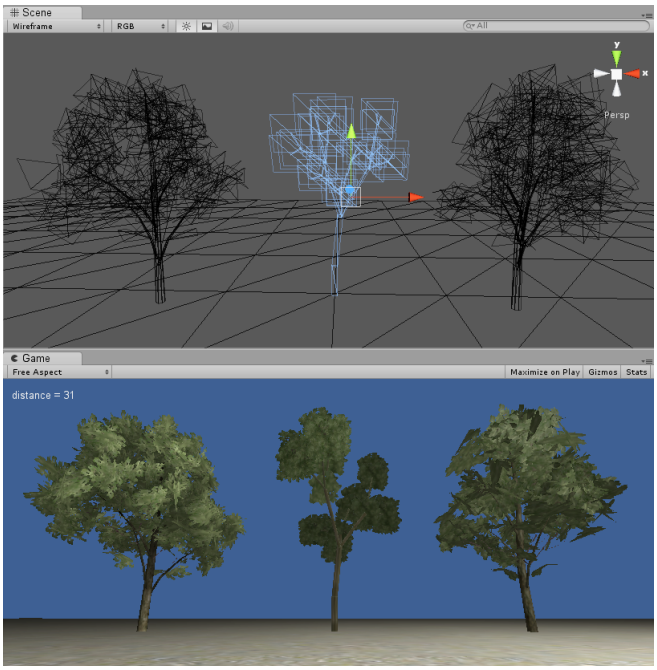

Figure 13: *Comparison between a stock shaded tree (left), our simple approach (center), and the same stock unshaded tree (right).*

Figure 14: *Close-up view of generated tree.*

Figure 15: *Single texture used for all 3D rendered trees.*

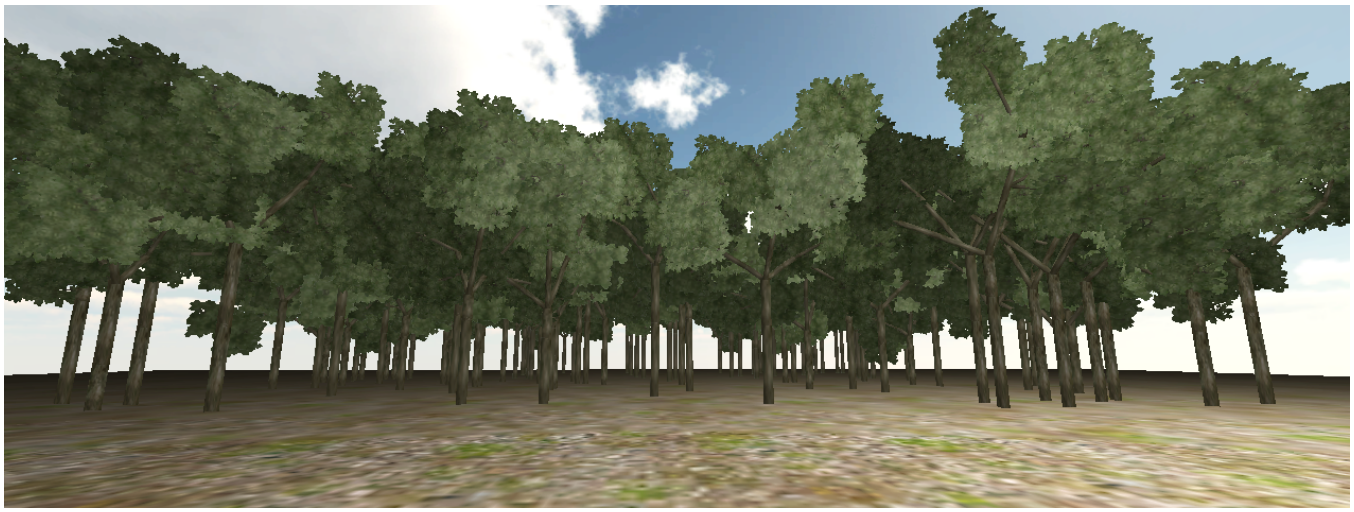

Figure 16: *Forest comprised of 100 trees, resulting in about 18,600 triangles and 35,500 vertices.*

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 (Sept.), 507–516.

DAY, A., AND WILLMOTT, J. 2005. Compound textures for dynamic impostor rendering. *Computers & Graphics 29*, 1 (Feb.), 109–124.

DÉCORET, X., DURAND, F., SILLION, F. X., AND DORSEY, J. 2003. Billboard clouds for extreme model simplification. *ACM Transactions on Graphics 22*, 3 (July), 689–696.

DENG, Q., ZHANG, X., YANG, G., AND JAEGER, M. 2010. Multiresolution foliage for forest rendering. *Comput. Animat. Virtual Worlds 21* (January), 1–23.

FEDERL, P., KARWOWSKI, R., MECH, R., AND PRUSINKIEWICZ, P. 2003. L-systems and beyond. *Course Notes (SIGGRAPH 2003)*.

GARDNER, G. Y. 1984. Simulation of natural scenes using textured quadric surfaces. In *Computer Graphics (Proceedings of SIGGRAPH 84)*, 11–20.

GILET, G., MEYER, A., AND NEYRET, F. 2005. Point-based rendering of trees. In *Eurographics Workshop on Natural Phenomena*, 67–72.

GUMBAU, J., CHOVER, M., REMOLAR, I., AND REBOLLO, C. 2010. View-dependent pruning for real-time rendering of trees. *Computers & Graphics In Press, Accepted Manuscript*, –.

LACEWELL, J. D., EDWARDS, D., SHIRLEY, P. S., AND THOMPSON, W. B. 2006. Stochastic billboard clouds for interactive foliage rendering. *Journal of Graphics Tools 11*, 1, 1–12.

LINDENMAYER, A. 1968. Mathematical models for cellular interactions in development i. filaments with one-sided inputs. In *Journal of Theoretical Biology*, vol. 18, 280 – 299.

LIVNY, Y., PIRK, S., CHENG, Z., YAN, F., DEUSSEN, O., COHEN-OR, D., AND CHEN, B. 2011. Texture-lobes for tree modelling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2011)*.

LLUCH, J., CAMAHORT, E., AND VIVÓ, R. 2004. An image-based multiresolution model for interactive foliage rendering. In *WSCG*, 507–514.

MAX, N., AND OHSAKI, K. 1995. Rendering trees from precomputed z-buffer views. In *Eurographics Rendering Workshop 1995*, 74–81.

MEYER, A., NEYRET, F., AND POULIN, P. 2001. Interactive rendering of trees with shading and shadows. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, 183–196.

OPPENHEIMER, P. E. 1986. Real time design and animation of fractal plants and trees. In *Computer Graphics (Proceedings of SIGGRAPH 86)*, 55–64.

PARISH, Y. I. H., AND MÜLLER, P. 2001. Procedural modeling of cities. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 301–308.

PRUSINKIEWICZ, P., LINDENMAYER, A., AND HANAN, J. 1988. Developmental models of herbaceous plants for computer imagery purposes. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, 141–150.

PRUSINKIEWICZ, P., , HANAN, J., HANANZ, J., HAMMEL, M., HAMMELY, M., AND MECH, R. 1996. L-systems: From the theory to visual models of plants. *Plants to ecosystems: Advances in computational life sciences I*.

PRUSINKIEWICZ, P., SAMAVATI, F., SAMAVATI, F., SMITH, C., SMITH, C., KARWOWSKI, R., AND KARWOWSKI, R. 2002. L-system description of subdivision curves. In *International Journal of Shape Modeling*, vol. 9, 41–59.

QIN, X., NAKAMAE, E., TADAMURA, K., AND NAGAI, Y. 2003. Fast photo-realistic rendering of trees in daylight. *Computer Graphics Forum 22*, 3 (Sept.), 243–252.

REEVES, W. T., AND BLAU, R. 1985. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *Computer Graphics (Proceedings of SIGGRAPH 85)*, 313–322.

SHADE, J., LISCHINSKI, D., SALESIN, D. H., DEROSE, T. D., AND SNYDER, J. 1996. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, 75–82.

SILLION, F. X., DRETTAKIS, G., AND BODELET, B. 1997. Efficient impostor manipulation for real-time visualization of urban scenery. *Computer Graphics Forum 16*, 3 (Aug.), 207–218.

SZIJARTO, G., AND KOLOSZAR, J. 2004. Real-time hardware accelerated rendering of forests at human scale. In *Twelfth International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (Winter School on Computer Graphics)*, 443–450.

UNITY, 2011. `http://www.unity3d.com/`.

WEBER, J., AND PENN, J. 1995. Creation and rendering of realistic trees. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, 119–128.