Rendering Dense Foliage with Stochastic Simplification

Abstract

In this project, I explore the challenges of rendering scenes consisting of many high-detail foliage models in a modern game engine. Focusing on the issue of popping artifacts caused by standard level-of-detail techniques, I show how stochastic simplification techniques can be applied to generate smooth level-of-detail transitions and scale performance across different scenes and hardware. I have implemented a simple rendering engine which demonstrates this technique to render a scene with large trees.

Background

Realistic foliage rendering remains a difficult task for modern game engines primarily due to the high level of geometric detail needed to define a tree or plant mesh with individual leaves. Due to hardware limitations, games have historically avoided rendering complete foliage meshes: At first, simple polygonal models approximating the outline of a tree or shrub were used instead (this technique is still common in less 'realistic' games). Later, games began using billboard techniques, wherein a tree would be rendered as a collection of intersecting 2D planes with transparent textures, an effect resembling a cardboard cutout with overlapping sections at different angles to give the illusion of depth. While billboards are efficient and convincing from a distance (far enough that small camera movements do not betray the 'flatness' of the foliage), they are quite unsatisfactory when examined close-up.

With the increasing computational power and vertex-throughput of newer GPU's, game developers have begun using more detailed foliage meshes, including individual leaves. This creates a problem as even the most powerful hardware cannot handle a large scene with many models, although it is reasonable to render a few very high-detail meshes. To reduce scene complexity to a manageable polygon count, developers have turned to level-of-detail techniques already common in games. These techniques function on the principle that distant objects can be rendered with less geometric detail than the same object viewed up-close, because they will cover fewer pixels on screen. The basic process is to generate several (usually 3-5) different meshes at various levels-of-detail (LODs) and choose which mesh to render based on some function of the camera distance. For distant meshes, a LOD can be generated that ignores or filters high-frequency geometric detail that would be missed by screen-space uniform coverage sampling.

While LOD techniques are a powerful tool for simple meshes, they perform poorly for dense foliage meshes due primarily to popping artifacts. An object 'pops' on screen whenever a small camera movement causes a change in the selected LOD which involves a very large or visible change in the object. Preventing popping artifacts can be done either by cleverly obscuring objects at transition points or by generating LODs which attempt to minimize the visible geometric differences; the former is not always possible, and the latter is in direct contrast with the goal of LOD techniques. Standard techniques for foliage involve pre-computing a few versions of the mesh, generated to approximate the correct screen coverage and visible color at a certain distance, with a set of billboards for extremely far distances; the result is good visual fidelity over selected camera regions, with dramatic popping artifacts at boundaries.

The use of a few select LODs also creates problems of performance because many meshes cannot be placed in proximity, or there will be overlapping regions at which dense meshes are used for many objects exceeding hardware limitations. Game developers must carefully position scene elements to avoid creating dead spots where performance drops, and carefully tune LOD-selection algorithms to maximize visual fidelity over scenes with varying amounts of visible geometry. That is, individually computing LODs based on distance may not be sufficient, or may require conservative algorithms that do not yield pleasing results for a single high-detail mesh.

Alternative - Dynamic LODs

An alternative technique exists in film rendering, where high-detail meshes are frequently used in long rendering jobs, and the limiting factor on total scene complexity is available memory not polygon throughput. **Cook et al.** describe how aggregate detail from foliage meshes can be reasonably approximated via stochastic simplification; essentially, their method renders foliage meshes by selectively rendering a random subset of the plant's leaves, with enough detail to achieve visual fidelity based on screen coverage. I propose that stochastic simplification can be applied at run-time in a game engine to

dynamically generate LOD meshes for any camera position and scene complexity. By continuously varying mesh geometry at the level of individual leaves, we can create smooth transitions that avoid popping artifacts; the level of detail can also be determined by a combination of scene complexity and hardware considerations, allowing better visual quality over a wider range of hardware and scene detail than can be achieved with a small set of LODs.

Basic Approach

The goal of our algorithm is to generate, for a given mesh at position P with N leaves and a camera position C, a renderable mesh which reasonably approximates the result of rendering the full N-leaf mesh. In our implementation we assume that leaves are geometrically identical in local space, so that instancing can be used to efficiently render many copies of the same geometry at unique locations and orientations, but this is not required for our approach as long as all leaves have the same number of vertices. The algorithm itself should be as simple as possible so that it can be performed at run-time and updated every frame; moreover the meshes generated at very similar camera positions should be similar enough to avoid popping artifacts.

Following **Cook et al.**'s approach, slightly simplified for efficiency, we compute a ratio $0 \leq \lambda \leq 1$ from the distance to the object. Using a single base mesh with the leaves pre-sorted in a random order, we render the first λN leaves, updating λ each frame. Here, we observe that a random subset is sufficient to approximate the full mesh given the proper value of λ, and that as the camera moves towards or away from the object, λ will increase or decrease steadily, causing leaves to be added or removed in small steps, even one-at-a-time. Our formula for computing λ is similar to that of **Cook et al.**: Given an artist-selected distance $0 \leq d_0 < \infty$ at which simplification begins, and a rate of simplification $0 < h \leq \frac{1}{2}$, we compute the distance $d = ||P - C||$ and set

$$\lambda = min\left((\frac{d0}{d})^{\log_h(\frac{1}{2})}, 1\right)$$

We note that this simple technique, with proper tuning, can generate minimally detailed meshes to achieve maximal visual fidelity at any camera distance from a single unmodified base mesh. No pre-computation or unique LOD meshes are necessary, and animated meshes can be used without generating unique animations for lower LODs.

Refinement - Leaf Scaling

While the basic approach outlined above is sufficient to generate LODs for a single mesh, we still observed popping artifacts from individual leaves. Some steps can be taken to address this, such as prioritizing the most likely obscured leaves near the center of a leaf cluster to eliminate first, but before attempting any such mesh-specific techniques we first turn to a general approach, again from **Cook et al.** The original researchers at Pixar noted that it was necessary to increase the size of remaining leaves after simplification, in order to correct for lost area in screen coverage. We apply their technique in our runtime implementation, scaling the leaves by a factor of $s = \frac{1}{\lambda}$. In addition, we tested **Cook et al.**'s first approach to 'smooth animation' by adjusting the size of the last few leaves rendered so that they appear to grow or shrink rather than appearing and disappearing suddenly. Instead of rendering all λN leaves at scale s, we render the first 95% of leaves at this scale, and the remaining 5% on a linearly decreasing scale from s to 0. As a result, leaves do not appear suddenly, but grow into position as λ increases, and shrink to nothing as λ decreases.

Obviously this effect can still be distracting, and it is questionable how well it might work. The Pixar implementation relied on applying simplification only when the distance was great enough that the size of a leaf on screen was less than one pixel; thus, coverage could be preserved by scaling leaves to ensure that sub-pixel samples remained covered, but leaf edges would never visibly move. In our tests, however, we found that scaling could be applied even at very close distances, and was neither distracting nor even detectable for basic meshes. The reader may wish to view our videos or experiment with the code to confirm this; it is our belief that scaling is sufficient for smooth transitions on meshes with reasonably small leaves, as long as the base distance and simplification rate are not overly aggressive.

Basic Results and Open Questions

From my limited tests, I found that stochastic simplification with leaf scaling yielded very smooth, natural transitions, given sufficiently powerful hardware to render the highest LOD mesh. I have created a simple rendering engine using very basic object culling and lighting methods, to ensure that the primary bottleneck is vertex throughput, and found that significant performance gains can

be achieved across moderate camera distances. In general, framerate increases roughly linearly as leaf count decreases, as one would expect, but my tests were offset by the need to render a full-detail tree trunk/branch mesh. I suspect that simplification can be made more aggressive by dividing leaf meshes into segments or clusters of nearby leaves and further simplifying the farthest or occluded clusters, as well as by weighting leaf distributions based on their distance from the center of the tree: We only simplify meshes at a distance, i.e. when the camera is not located within the tree branches, so the camera must be looking in toward the tree, and the center-most leaves are less likely to be noticed disappearing (the backside leaves moreso, but we want a single sorting order which works for any viewing direction). However, I have not implemented more sophisticated techniques for individual meshes, as this is a problem that can best be tuned for the particular mesh in question.

More importantly, I have not yet experimented with contrast preservation as described by **Cook et al.**; this is noticeable as a slight brightening of trees in the distance under aggressive simplification, and could likely be corrected to within tolerance by a simple gamma reduction. **Cook et al.** also noted that scaling by $\frac{1}{\lambda}$ breaks down at extremely small values of λ, and this effect is very noticeable in my implementation. To handle this, I impose a requirement that $\lambda \geq 0.005$, and do not render meshes for which a smaller λ would be selected. In practice, this is the range at which performance gains from stochastic simplification are no longer meaningful, since a few thousand leaves may still be rendered to cover a handful of pixels on screen. I propose that a full renderer continue to use billboards at extremely far distances: Simplification at this range generates large leaves covering most of the area of the tree for a very similar effect; if precomputed billboards are not desirable, they can likely be generated by a single frame of rendering with stochastic simplification, and re-used until λ increases above a threshold or the viewing angle changes significantly.

Extension - Handling Complex Scenes

While stochastic simplification yields promising results for individual models, the performance gains are not generally sufficient for scenes with many visible models. In this section, I assume a hardware setup with sufficient throughput to render a few high-detail meshes at full resolution, and demonstrate how stochastic simplification can be extended to dynamically fit scene complexity to hardware limitations. Several possibilities exist for more aggressively simplifying meshes that are partially obscured by other objects; for instance, one might use raycasting against a bounding volume hierarchy, or a low-resolution rasterization into tiled lists, to determine self-occlusion between leaf clusters, and further reduce leaf counts for clusters toward the rear. However, each such technique comes at its own cost and has significant failure cases, namely view positions that scan the whole scene but without many models obscuring each other, such as a low-altitude fly-by of a forest.

My goal instead was to devise a simple technique for aggressively tuning all λ values for visible leaf meshes to fit the overall scene complexity within performance constraints. Here, I make a key observation about the use of LOD techniques in games as opposed to film: In film rendering, visual fidelity is always of the utmost priority, and LOD techniques are limited to the maximum simplification that can be performed *without* changing quality of a single model. In games, the goal is a smooth, immersive experience, and the player's attention is usually focused on specific objects, often in the foreground. As a result, background detail can often be sacrificed in exchange for high-quality rendering of nearby objects. Games often apply this technique aggressively, either by distracting the player with nearby effects or more blatantly using 'fog' or a short horizon to block distant objects.

In my experiments, I noted that for distant meshes, extremely high levels of simplification can be applied without the player noting that a tree mesh has gone 'past' the point of visual correctness; indeed, it is difficult to determine this point with the naked eye when leaves are very small. Arguably the entire technique rests on this assumption - that players cannot tell the difference between a tree with $80,000$ leaves and a tree with $50,000$ leaves as long as the overall screen coverage is similar and they do not notice the *transition* between leaf counts. (Note that **Cook et al.** take no chances - their film rendering implementation performs simplification only when leaves are too small to be counted.)

As a result, it is possible to set λ values for distant meshes much lower than normally determined by the basic techniques above, as long as the transition from low to high λ values remains smooth and the curve is not so steep that a player notices the difference when moving through the scene. This yields a simple approach for modulating λ values across a scene to budget overall performance.

Extended Simplification Algorithm

Given a set of n visible models at respective distances $d_1, \cdots, d_n$ with resulting (non-zero) simplification factors $\lambda_1, \cdots, \lambda_n$ our goal is to assign to each model a new value $0 < \lambda_i' \leq \lambda_i$ such that the closest models to the camera retain the highest values of λ and the furthest models are simplified more aggressively. This should be determined by distance relative to the camera, not by sorting the models, so that adjacent models at the same distance receive similar values. Moreover, the rate of reduction should be controlled by overall scene complexity - if very little geometry is visible, then we should render it at high resolution, while a camera frustum that captures much of the scene geometry *requires* further simplification to fit within frame time limits. Lastly, the overall difference in reduction of λ for close and far models should be controlled by the difference in distance, so that a narrow band of models does not cause a large reduction in mesh detail over a very small distance delta (this would break our requirement that mesh transitions are smooth and not too fast to be believed).

In my implementation, I loosely approximate the overall scene complexity using the model count n, and allow a scene-specific value m defined to be the number of models at which roughly linear falloff in λ_i' with distance is desirable, as well as a maximum distance value z at which most models should reduce to a few pixels and be culled or replaced with billboards. In my tests, I set z to be the distance to the far clipping plane, and use an extremely long view frustum when performance allows. This displays the adaptability of stochastic simplification to large scenes, as well as the continued need for billboards to save on geometry at extremely far distances, and the problem with scaling and overly aggressive simplification rates when the clip distance is set much closer than the distance at which projected models are too small to see. Given these values, I compute λ_i for all visible models as follows:

$$d_{min} = min_i\left(d_i\right), \qquad d_{max} = max_i\left(d_i\right)$$

$$d_{range} = d_{max} - d_{min}$$

$$w = \frac{d_{range}}{z}$$

$$p = \frac{m}{n}$$

$$\lambda_i' = \lambda_i * \left(1 - a * w * \left(\frac{d_i - d_{min}}{d_{range}}\right)^n\right)$$

Here $0 \leq a \leq 1$ is a constant factor meant to prevent λ from reducing to zero at the far clipping plane; if z is accurately set to a distance large enough that all models projected to the screen are smaller than one pixel, then in theory $a = 1$ could be used, but in practice this can be problematic due to computations with extremely large floating point numbers. Instead, it is more effective to set z to a smaller value (such as the far clipping plane, which still provides a constant limit on scene complexity for most games) and choose a value of a which limits further reduction to reasonable values. In my implementation I used $a = 0.5$ for all scenes, but further experimentation with distinct values of a and z might yield a more meaningful selection criteria than pure experimentation.

Conclusion and Extended Results

Using my extended simplification technique, I was able to render large scenes composed of many scattered models at interactive rates. Framerate on extremely complex scenes was not ideal even on a high-end machine ($\sim$ 10 fps for the forest scene with 2x nVidia GeForce GTX 780), but here any reader should note that my implementation is still hampered by rendering full-detail branch meshes, and in any case the farthest meshes could likely be convincingly approximated with billboards. However, with only very basic culling and the techniques outlined above I can achieve a high-speed, interactive fly-by of a relatively scattered forest scene, using no additional LOD meshes or pre-computation other than shuffling the leaves. The rendering is smooth with almost no noticeable transitions, and the player can approach individual trees at will to inspect them in full detail.

My approach does not, however, perform sufficiently well for extremely dense forest scenes. The most problematic case is a large group of trees, roughly the same distance away from the camera but all very close; my implementation detects this is a case where there is sufficient detail to merit aggressive simplification, but additional reduction cannot be applied since all the detail is clustered close to the camera. Fortunately, this is a case where significant gains can be made by occlusion-based simplification. Further investigation is required into techniques that take into account occlusion from nearby meshes when performing simplification, so that performance budgeting can be adjusted for this scenario.

Lacking this, I believe that my general implementation shows promising results for stochastic simplification as a dynamic approach to LOD generation for high-detail meshes. Like tessellation, this is a technique which dynamically scales geometric complexity to allow smooth transition between simple LOD's and very high detail foreground meshes without popping artifacts. Moreover, the scene- and object- specific values for simplification rate computations can be adjusted to fit the performance budget of any device, allowing complex scenes to be mapped to a wide range of hardware limitations by more aggressively reducing distant objects when necessary. Unlike traditional LOD's, dynamic simplification naturally avoids creating areas where multiple objects 'pop' simultaneously, since all leaf meshes make small, roughly continuous transitions. The computed λ values can be further adjusted to avoid dead zones with low detail or peaks with too many high-detail objects, by scaling based on additional scene factors. They need not be identical for all camera positions at the same distance from an object, and the difference can be hidden from the player simply by ensuring that λ varies smoothly toward the same result whenever the camera approaches the same position in space from a different direction.

The main obstacle to adoption of techniques involving stochastic simplification, besides the open issues mentioned earlier, is the simple need for increased vertex throughput in commonplace graphics hardware. The desire for increased vertex rates and the ability to handle many very small triangles has already been expressed by game developers hoping to make better use of tessellation hardware; I believe that stochastic simplification and tessellation have similar motivating factors and that the former will see increased interest as hardware continues to improve alongside continued research in shading small triangles.

References

Robert L. Cook, John Halstead, Maxwell Planck, and David Ryu. 2007. Stochastic simplification of aggregate detail. ACM Trans. Graph. 26, 3, Article 79 (July 2007).

Other Links

The website for this project, including source code and sample scenes, is available at

`http://fgomezfr.github.io/foliage/`

A few sample videos demonstrating the basic scenes and some additional LOD tests are available here as of 12/14/2014.