Table of Contents

Real-Time Modeling

The tree model depicted to the right was processed with the 3ds Max processor script, then rendered with mental ray in Autodesk®'s 3ds Max®.

See the sections below for things to consider above and beyond just modeling the tree when creating models for games or other real-time applications.

Only the SpeedTree for Games license permits SpeedTree's use in real-time applications. Contact for more information.

Efficient Real-Time Trees

Library trees designed for real-time use are signified by an "_RT" suffix on the filename. These trees generally have 1,000 - 12,000 triangles at the highest LOD, and significantly less at lower LOD. If you are making trees for an interactive project such as a video game or other visual simulation, see the following sections for useful tips:

1. **Level of detail** - SpeedTrees smoothly transition from medium to low detail models depending on their distance from the camera. Tune these settings to get an efficient LOD transition. *More info: Level of Detail*

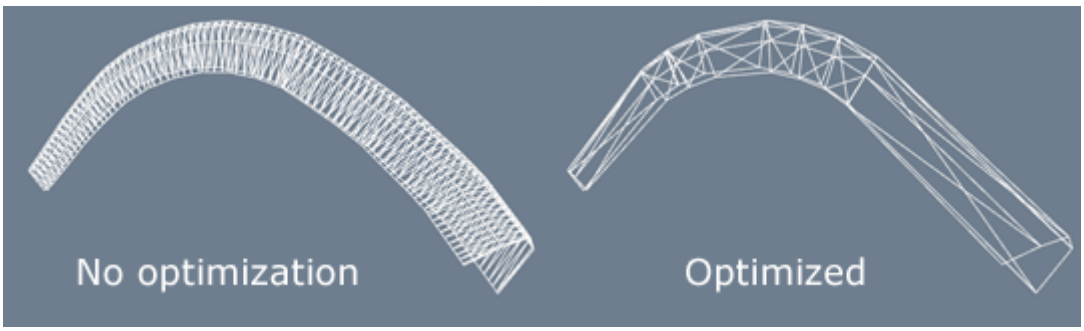
2. **Leaf collision** - Leaves are the slowest part of the tree to render. Use leaf collision to remove unnecessary leaves.

More info: Leaf Collision

3. **Wind** - Take advantage of SpeedTree's built-in wind effects. Using settings available in the Modeler and Compiler, wind effects can range from inexpensive but effective to detailed and realistic, appropriate for detailed hero trees and work with the SpeedTree SDK as outlined in the SpeedTree Pipeline.

More info: Wind

4. **Segment optimization** - Use the built-in Optimization features to get rid of unnecessary branch and frond length segments (there is a specific Frond optimization property for additional frond optimization of frond geometry). Optimization takes the spine angle into account, so segments are added and removed adaptively based on the curvature of the branch (see below). Optimization can also be increased for lower LOD states.



5. **Reduce the number of draw calls** - Each “draw call” requires its own pass on the tree when rendered through the SDK. The more draw calls, the slower it will be to render the tree. Generally, most trees require 3 draw calls – one for each major geometry type (branches, fronds, and leaves), although 2 calls can generally be achieved with a little planning. Any tree requiring more than 3 draw calls should be considered a special case.

More info: Reducing the Number of Draw Calls

6. **Reduce overdraw** - “overdraw” is when more than one triangle face overlap on screen. The extra triangles slow down computation, but don't add anything visually.

More info: Reducing Overdraw

7. **Always create a texture atlas when compiling trees** - It sounds like common sense, but provided the importance of draw call count, using a texture atlas is vital for best performance. Atlasing will combine all leaf, frond, and cap textures into only a single texture lookup.

More info: Know your target platform. Talk to your engine programmers about which is more optimal for your platform, smaller atlases (one for each tree model), or larger atlases that combine textures from multiple tree models.

Reducing The Number of Draw Calls

Each “draw call” requires its own pass on the tree when rendered through the SDK. The more draw calls, the slower it will be to render the tree. Generally, most trees require 3 draw calls – one for each major geometry type (branches, fronds, and leaves). Any tree requiring more than 3 draw calls should be considered a special case.

What Counts As a Draw Call

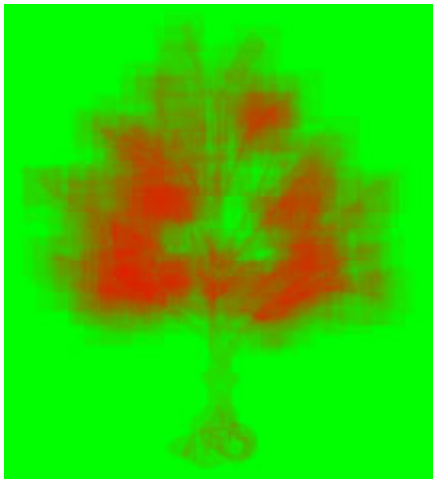
A draw call occurs for each “color set + texture lookup” per geometry type. Any time one of the geometry types have to change either color sets OR material, a new draw call is issued. Keep in mind that usually all leaves, fronds, and caps are contained within a composite atlas texture that gets created by the SpeedTree Compiler during the compilation process.

This means that if your tree has two different leaf “materials” (in the Modeler), provided that those two leaves use the same color set, they will amount to only one draw call in the SDK since they will share both a color set and texture atlas once compiled.

Likewise, we follow a similar path of merging draw calls together in the case of cap geometry. “Caps” are actually a fourth geometry type. However, to avoid a fourth draw call, the caps and fronds can easily share both a color set and texture set once compiled. Do keep in mind that this optimization hinges on them sharing a color set and texture atlas. The same optimization can be used again in the case of embedded mesh forces, which can be combined with the leaves draw call, exactly like the caps and fronds.

While leaves, fronds, and caps all get put into the atlas texture, branch textures do not, since they tile. As is the case, every material assigned to branches require a separate draw call. **Only use multiple branch materials in very special cases**, such as for a “hero” tree.

Reducing Overdraw



“Overdraw” is when more than one triangle face overlap on screen. Since theoretically only one of these overlapping triangles contribute to the final pixels on-screen, the other triangles can be thought of as superfluous. Of course, some overdraw will always occur. However, there are practical

measures that can be taken to reduce overdraw as much as possible:

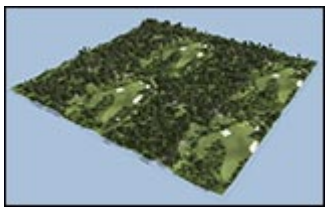
1. Enable backface culling on branch color sets. You will need to ensure that backface culling is enabled on all color sets assigned to branch geometry. Otherwise, the backfaces will contribute to the overdraw.

2. Cut out transparent pixels from leaf and frond meshes. Fronds, or rather “image-based branches” rendered as fronds usually contain a high percentage of transparent pixels. The underlying geometry still contributes to overdraw, even if you can't see the triangles that comprise it. Instead of using standard fronds, use a mesh frond with the transparent pixels cut out, as much as is reasonable (there will be a trade off between the tightness of the cutout and the number of vertices required to form the shape out of geometry).

More Info: See this section for more info on making mesh cutouts.

3. Use the “overdraw visualization” rendering mode for diagnosis. This rendering mode will give you a “heat map” of overdraw in your model. The pixels that are the most red have the highest number of triangles underneath them. An ideal tree would have as little red as possible.

World Building



“World building”, or the practice of placing all of your trees in the scene, can be a tedious job. World building tools have been incorporated into the SpeedTree toolset. Proxy objects are placed on a terrain, and their locations, scale, and rotation are exported in an ASCII format called SWA (SpeedTree World-Building ASCII) files. This text file can be read in by other apps, and instances of real trees can be put in the correct locations. See the following sections for further details:

1. World Building Tutorial
2. Zones
3. Proxies
4. Exporting SWA Files

- Site Map
-
- Questions?
- 
- 
- 
-