Real-time Rendering of Photorealistic Natural Tree

Boris J. Wang[1, 2]

1. Institute of Computing Technology, Chinese Academy of Sciences

2. Graduated School of Chinese Academy of Sciences (Wang Jia Ping 200228013202862)

Abstract

This paper presents a method for rendering photorealistic tree in real-time based on fractal idea. A branch glowing algorithm is introduced for gradually modeling the tree. This algorithm also provides the capability of controlling shape of the tree in a parameterized way. The demonstration program gives the performance of this method. It renders scene 15-40 frames per-second on moderate PCs.

Introduction

Rendering natural objects is a difficult task for geometry-based rendering because many natural objects have very irregular surface. Modeling Irregular surface or structure is a hard work, very time/memory consuming, in BREP paradigm. In another side, based on fractal idea, most chaotic phenomena can be described in fractal means. A fractal-based approach is made for modeling natural tree.

Photorealistic rendering is another challenge in computer graphics. Most photorealistic scenes are generated by accurate but slow method of Ray Tracing. For tradeoff with rendering performance, Ground Shading, Texturing, Z-Buffering and alpha blending is used in this paper, instead of Ray Tracing, for meeting speed requirement of real-time application.

Modeling the tree

Figure 1 shows the basic idea of my method. A tree is composed by branches and leaves and a branch consists several segments connected together. A tree is recursively defined as the following:

1. A Tree is a branch.
2. A branch is a segment connecting to one or two descendent branches.
3. Leaf appears at the tip of every branch.

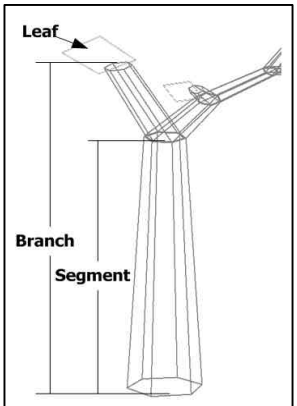

Figure 1. A Tree

Primitives of a tree

A **Segment** is a cone, which is approximated by six side-quads. The shape of all segments is parameterized by four factors: Twist, Expand, Length and Radius. Figure 2 shows the meaning of them. Twist and Expand is values of angle, which specifies the orientation of the segment. The two parameters control global distortion of branches. Radius altogether with Length controls the size of it.

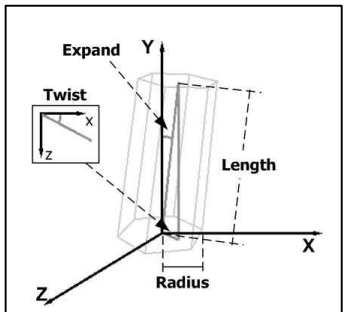

Figure 2. Segment Model

A **Leaf** represented by just one quad. Irregular border of leaf is visualized via Alpha Testing. It is also parameterized for different appearance. Showed as figure 3, Scale controls the size of it. Expand and Twist, like the parameters of the same name in figure 2, represents orientation of the leaf.

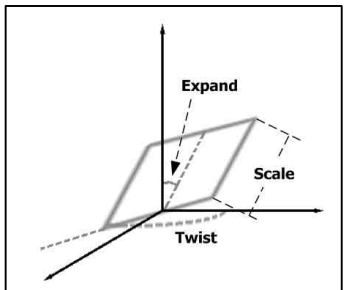

Figure 3. Leaf Model

Glowing the tree

A Branch is generated by glowing segments. Given a root segment B_0 , the four-parameter of descendent segments B_n are calculated as the following:

$$\begin{cases} Twist_{B_n} = Twist_{B_{n-1}} + Twist_{Global} \\ Expand_{B_n} = Expand_{Global} \\ Length_{B_n} = Length_{B_{n-1}} \bullet Length_{Global} \\ Radius_{B_n} = Radius_{B_{n-1}} \bullet Radius_{Global} \\ Depth_{B_n} = Depth_{B_{n-1}} - 1 \end{cases} \qquad n \in (1, Depth_{B_0})$$

Glowing stops at B_n when $Depth_{Bn}=0$.

All symbols with subscript of *Global* are constant of global parameters that affect the final result of the tree modeling. Every

next segment is generated according to the formula above and is placed on the smaller end of the previous one. Properly placing descendent segment is done by local coordinate space. When assembling a new segment, a local coordinate space is generated based on the four-parameter of the previous segment. Thus, the current segment and all its descendents are modeled based on the new local coordinate space. Shifting to the new local coordinate space is implemented by the following coordinate transform.

$$\begin{bmatrix} 1 & 0 & 0 & R \\ 0 & 1 & 0 & L \\ 0 & 0 & 1 & R \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} Cos(T) & 0 & -Sin(T) & 0 \\ 0 & 1 & 0 & 0 \\ Sin(T) & 0 & Cos(T) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & Cos(E) & -Sin(E) & 0 \\ 0 & Sin(E) & Cos(E) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bullet \begin{bmatrix} Cos(T) & 0 & Sin(T) & 0 \\ 0 & 1 & 0 & 0 \\ -Sin(T) & 0 & Cos(T) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This transformation matrix is expressed using normalized coordinate. Symbol T E L R represents Twist Expand Length and Radius.

While segments are generated recursively, all previous local coordinate transformations are accumulated. The depth of the recursion is the global constant $Depth_{Global}$.

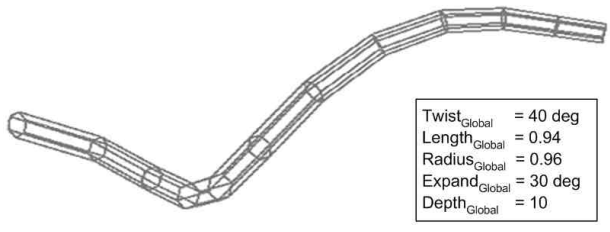

Figure 4. A branch consists of ten segments

The shape of natural tree is complex and various. However, the fractal view provides a feasible solution for approximation. In fractal view, a branch can be regard as a minor version of the whole tree, which is called self-similarity. Complex structure can be described or defined by properly duplicating basic structure element. Based on this idea, A Tree is defined as a primary branch, the trunk, and all its possible descendents. The basic structure element is defined as a base segment with two bifurcated descendent branches, showed as figure 5.

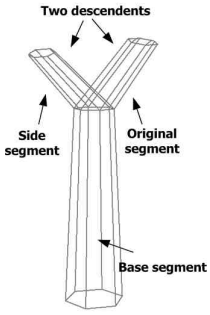

Figure 5. Basic structure

The two descendents is not treated equally. One is the original one as a normal descendent segment that is generated in the way of glowing segments, which is early mentioned in the initial part of this section. Another is called Side-Segment, which will be regard as the root segment of a new branch, the side-branch, in further glowing. Four-parameter of side-segment is calculated as:

$$\begin{cases} Twist_{Side} = Twist_{Base} + Twist_{Global} + 180\,\deg \\ Expand_{Side} = Bifurcate_{Global} - Expand_{Base} \\ Length_{Side} = Length_{Base} \bullet Length_{Global} \bullet SideScale_{Global} \\ Radius_{Side} = Radius_{Base} \bullet Radius_{Global} \bullet SideScale_{Global} \\ Depth_{Side} = \min(Depth_{Base} - 1, Depth_{Base} \bullet DepthScale_{Global}) \end{cases}$$

Comparing to the formula stated in glowing segment, three global parameters is added. As showed in figure 6, the two descendents bifurcated in the angle of $Bifurcate_{Global}$. The whole side branch is scaled at the ratio of $SideScale_{Global}$ and the depth of recursive glowing is reduced by $DepthScale_{Global}$. Treating descendents unequally add to the richness of the various shapes.

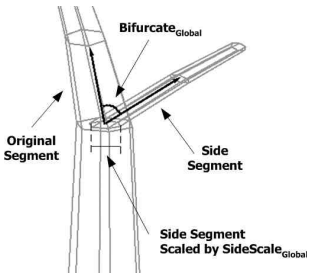

Figure 6. Side branch

Rendering the tree

In this paper, an OpenGL-based visualization is presented. On rendering, there are a lot of computer graphics programming details involved such as transformation, texturing, controlling normal, antialiasing, lighting and material settings. I just enumerate some key technology for the paper length.

Mipmapping

When rendering branches, texturing is utilized. But texture-mapping algorithm in OpenGL gives bad result at the tip of branches. Because pixels in texture image is selected, not blended, to fill the target surface when mapping bigger texture to tiny triangle. Mipmapping is introduced to overcome this problem. The basic idea of mipmapping is use smaller texture for tiny surface and bigger texture for large surface. Mipmapping is natively supported by OpenGL, which produces nice result under situation early mentioned. Figure 7 shows the different appearance of using mipmapping or not. At the large trunk and the near ground it seam no different but mipmapping greatly

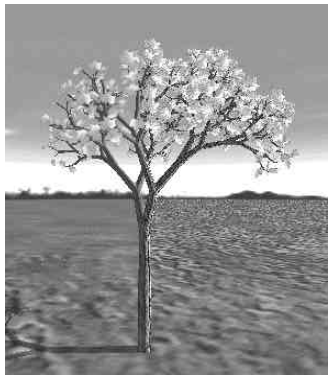

Figure 7. Mipmapping
Left part is rendered with mipmapping
Right part is rendered with normal texturing

improve visual quality at the branch tips and far ground. This technology is applied to rendering leaves and ground as well.

Alpha-Test

Figure 8. Rendering with Alpha-Test

Leaf have a very irregular edge, moreover there is a lot of leaves in a tree. It would totally kill rendering performance if the irregular edge of every leaf is constructed by mass of triangles. However most leaves are planar, thus we can cheat. I model the leaf as a quad and utilize Alpha-test to render leaves with its irregular edge. Alpha-test is based on texturing and also is mipmapping aware. It performs per-pixel judgment whether a pixel should be set or not based on alpha value associated with it, say the texture image contains alpha information, on texturing stage of rendering pipeline.

Shadowing

Rendering shadow of complex object is a big work in OpenGL because of no built-in universal shadowing mechanism provided. Many feasible solutions are implemented based on Stencil Buffer. Those methods do work but too complex. I provide a simple method for rendering accurate shadow to one plane. I render another tree in gray and paste it on the ground with blending. More detail, as showed in figure 9. I rotate reference frame about Y-axis and make Z-axis pointing to the direction of primary light. Then set scale ratio of Z to zero. Finally, rotate local coordinate space of the tree about X-axis making Z-axis upright against ground. With this transformation the tree is flattened along the lighting orientation and pasted on the ground. Figure 10 demonstrates the result of this shadowing model. Different shadow shape appears when tree changes. By the way, when render shadow, lighting, texturing and depth-test is disabled.

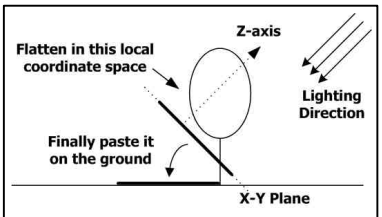

Figure 9. Shadowing Transform

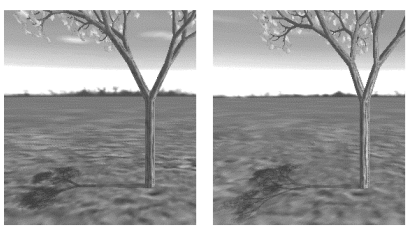

Figure 10. Shadows of different shape

Result and Demo Program

All early mentioned global parameters are exposed for interactive tuning. I write and test the demo program on a moderate laptop of 1.2G Celeron CPU with ATI Radeon M6 video card, 16M video memory on broad. It runs at 20-50 fps. Screen snapshot gives as the following: