White Paper

Fur (using Shells and Fins)

February 2007
WP-03021-001_v01

Document Change History

Version	Date	Responsible	Reason for Change
_v01	February 20, 2007	ST, TS	Initial release

Go to sdkfeedback@nvidia.com to provide feedback on Fur (using Shells and Fins).

Fur (using Shells and Fins)

Abstract

This sample illustrates the use of a new feature of DirectX 10, the Geometry Shader, to create new geometry on the fly. The sample uses the shells and fins technique to render fur on arbitrary triangle meshes. Fins are rendered by creating new geometry per frame along the silhouette edges of a mesh. Before DirectX 10 this required adding degenerate triangles to the mesh at every edge but now fins can be generated efficiently using the Geometry Shader.

Sarah Tariq
NVIDIA Corporation

Figure 1. Fur Rendered Using the Shells and Fins Technique

Motivation

Fur is typically rendered using one of two approaches; rendering all the individual hairs as geometry, or by methods inspired by volume rendering.

The brute force method of rendering the individual fur strands as pieces of geometry can create very realistic fur, and is especially preferable when the viewer is close to the fur or the fur is long. However, most models require millions of pieces of fur to look compelling, and rendering these is quite slow. Further more, rendering thin strands of fur causes serious aliasing issues; addressing these issues makes the rendering even slower, and not possible in real time.

Kajiya and Kay's volume rendering approach to fur is faster and avoids the aliasing artifacts present in rendering explicit geometry. Fur is represented as 3D textures containing density, orientation and lighting information for pre-generated fur. This volume is mapped onto a model and rendered by ray marching. In the past this technique had not been considered real time because of the long pixel shader with looping that is necessary for ray marching. With today's graphics hardware however, this technique might be feasible for certain models and situations.

The current standard method for interactive rendering of fur is another approach to volume rendering; shells and fins *[Lengyel01]*. This technique was used for example in the NVIDIA Wolfman demo, and is more tailored to the strengths of graphics hardware, utilizing texture mapping and alpha blending. 2D texture slices from a pre-generated 3D fur texture are used to texture concentric alpha blended shells that are extruded from the model (see *Shells* on page 3). These shells provide a reasonable approximation to fur in areas where the viewing direction is perpendicular to the surface, however, near silhouettes the shells become too transparent and their gaps become visible (see *Fins* on page 4). Fins (extra geometry extruded from the silhouettes) are used to alleviate this problem.

Prior to Direct3D10, in order to create fins the mesh would need to contain degenerate triangles on all edges. In Direct3D10 the geometry shader can be used to create these fins at only the edges where they are required, eliminating the need to bloat the vertex buffer with degenerate triangles. The fur textures are stored in a Texture Array—another new feature introduced in Direct3D10—eliminating the need to rebind a different texture for each shell pass.

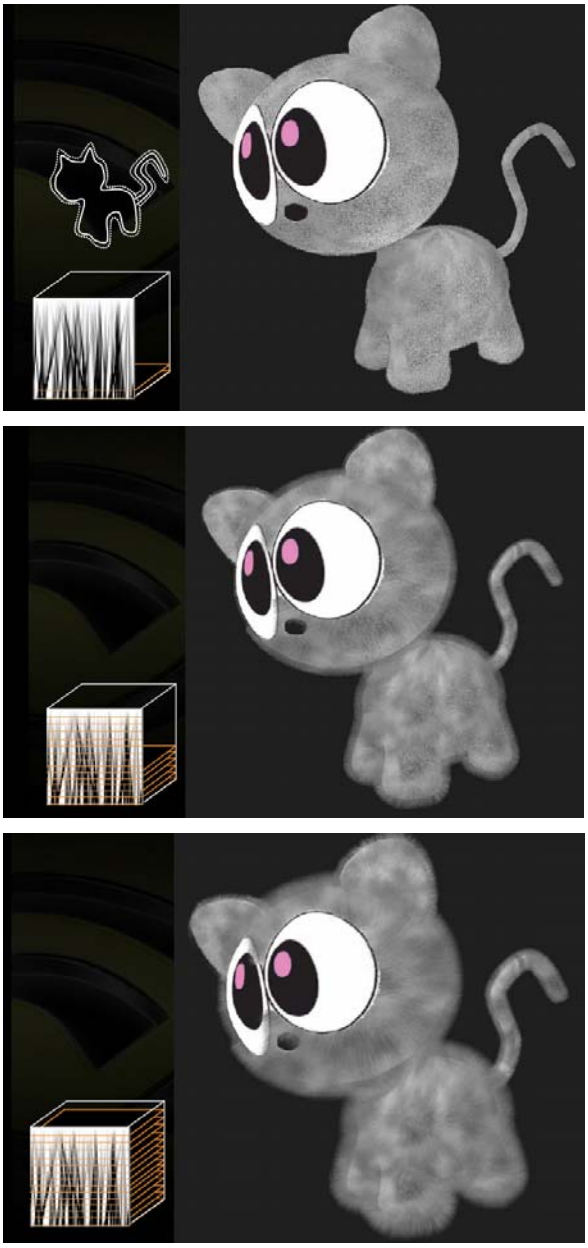
Technique

For completeness we briefly describe below the entire technique, but we encourage the reader to see *[Lengyel01]* or *[Sheppard04]* for details on rendering, lighting and texture generation as well as a discussion on tradeoffs.

Fur is rendered in several passes; first the mesh is rendered with depth testing and depth write, then the fins and finally the shells are blended on top. Fins are extruded from the silhouettes of the mesh and rendered with alpha blending. Finally the shells are rendered with alpha blending, one shell at a time, from the inside out.

Shells

Shells are rendered from the inside out by rendering the mesh **n** times and each time extruding the mesh along the normal by an amount proportional to the index of the shell. Each shell is textured with the appropriate texture from a Texture Array containing all the 2D texture slices, Figure 2. While rendering the shells, depth testing and alpha blending are enabled but depth writes are disabled.

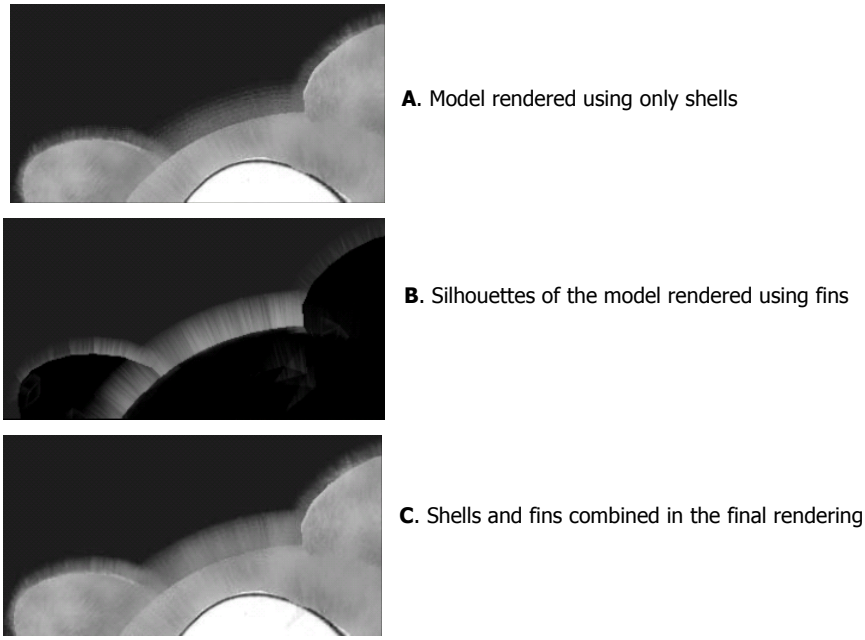


Shells are rendered by extruding the model outwards and texturing it with progressively higher slices from a 3D fur texture

Figure 2. Rendering Shells

Fins

Fins are rendered to preserve the illusion of fur along the silhouette edges, Figure 3.



A. Model rendered using only shells

B. Silhouettes of the model rendered using fins

C. Shells and fins combined in the final rendering

Figure 3. Adding Fins to the Silhouette of the Model

To render fins we process all edges of the mesh in the Geometry Shader and test each edge to see if it should be extruded.

To determine if an edge is a silhouette edge, we need to check that one of the triangles sharing the edge is facing the viewer and the other one is facing away. To determine if a triangle is facing the viewer we can take the dot product of its normal with the vector to the eye.

```
if dot( vectorToEye, triangleNormal ) > 0
    //triangle is facing the viewer
else
    //triangle is facing away from the viewer
```

To use this test we need to calculate the triangle normals for the two triangles sharing the edge. The line with Adjacency input primitive for the geometry shader can be used to input both the edge and its two opposite vertices which are used to calculate these normals:

```
//Geometry Shader for extruding the fins
[maxvertexcount(4)]
Void GSFins( lineadj VS_OUTPUT input[4], inout
TriangleStream<GS_OUTPUT_FINS> TriStream )
```

The geometry shader for extruding the fins takes as input four vertices in the **`lineadj`** format, illustrated in Figure 4A. From these vertices we can calculate the triangle normals of each of the two triangles. For example,

```
N1 = normalize(cross( input[0].Position - input[1].Position,
                      input[3].Position - input[1].Position ));
```

Edges that are on the silhouette are given by the test

```
if( eyeDotN1*eyeDotN2 < 0 )
```

as shown in Figure 4A. These are extruded into a fin (Figure 4B). This test checks that one of the triangles sharing the edge faces towards the viewer and the other faces away, the definition of a silhouette edge.

In addition to this, any edge that is almost a silhouette edge, defined by

```
if( abs(eyeDotN1) < finThreshold || abs(eyeDotN2) < finThreshold )
```

is also extruded as a fin to prevent fins from popping in and out during animation.

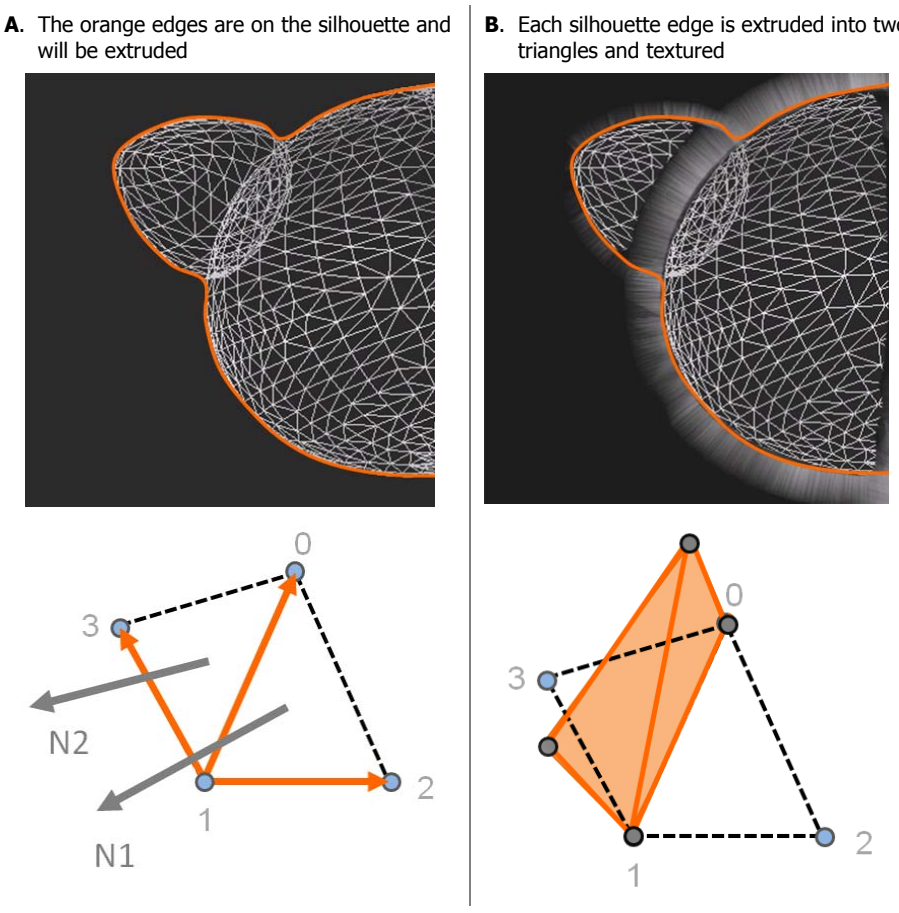


Figure 4. Detecting Silhouettes and Extruding Fins

Running the Sample

The sample uses the left mouse button to rotate the model and the mouse wheel to zoom. For comparison, checkboxes are provided to toggle off and on the rendering of the shells and fins. The sample also lets the user change the number of shells, the spacing of the shells, and the global comb direction of the fur. The right mouse button moves the comb vector, showed by the green arrow. This vector and the comb strength slider are used to define the influence of a global comb on the fur.

Integration

This technique requires the use of a line adjacency index buffer, which can be generated after loading the model. This buffer encodes each edge and its two opposite vertices as a lineAdjacency primitive that can be loaded in the geometry shader. The sample shows a simple way of generating such an index buffer.

The spacing of the shells is a parameter that depends on the size of the model, and requires some tweaking. *[Lengyel01]* recommend a spacing size of one thousandth of the model diameter which works well. To create the shorter hair on the tail we paint into the alpha channel of the mesh texture a modifier for the length of the fur. This modifier is multiplied by the shell increment to achieve spatially varying fur sizes. Areas with no fur, like the eyes and paws, have zero alpha, and are discarded in the pixel shader.

Finally, this technique requires pre-generated fur textures, which we have not covered. For an explanation of how to generate these textures using a particle system, refer to *[Lengyel01]* or *[Sheppard04]*.

References

- *Rendering Fur with Three Dimensional Textures*. James T. Kajiya and Timothy L. Kay. Proceedings of the 16th annual conference on Computer graphics and interactive techniques, 1989.
- *Real-Time Fur over Arbitrary Surfaces*. Jerome Lengyel, Emil Praun, Adam Finkelstein, Hugues Hoppe. Proceedings of the 2001 symposium on Interactive 3D graphics.
- *Real-Time Rendering of Fur*. Gary Sheppard, Honors Thesis, University of Sheffield. 2004. http://www.gamasutra.com/education/theses/20051028/sheppard_01.shtml
- NVIDIA GeForce4 Wolfman demo, 2002, http://www.nvidia.com/object/demo_wolfman.html

.

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com