5.9

Art-Based Rendering with Graftal Imposters

Joshua A. Doss, Advanced Visual Computing, Intel Corporation

joshua.a.doss@intel.com

Graftals are used to express the shape and formation of plants in a formal grammar for use in computer graphics. A close relative of fractals, *graftals* allow for a compact representation of foliage [Smith84]. Graftals have also been used in a non-photorealistic cartoon rendering implementation at interactive frame rates [Kowalski98]. Graftal imposters are used as a real-time method of drawing cartoon-style plants and fur using the geometry shader in modern GPUs. The particular style we aim to produce is inspired by Dr. Seuss's children's book illustrations [Seuss71].

An artist will provide sketches of graftal imposters along with a set of textures that place the foliage in a scene with a high amount of control over the final look and feel. The imposters are placed along the silhouette edges of the object. See Color Plate 12 for a full-color example.

Assets

Creating graftal imposters requires a set of assets in addition to the geometry—a texture atlas, control texture, and a vector field texture. The texture atlas contains three types of graftal imposters along with several variations of each type. A control texture provides information about what type of graftal imposter should be placed at a certain location on the mesh. The vector field gives a direction and the color texture provides information on the coloring of the landscape mesh as well as the graftal imposters.

Texture Atlas

The texture atlas contains the graftal imposter itself. It is created by specifying a few different types of graftal imposter types, in different rows. The exterior of each graftal imposter should have an RGB and alpha value of zero for all components. As you near the soft edge of the graftal imposter, the alpha value should go smoothly from zero to one in the middle of the stroke, giving you a smooth transition and reducing any aliasing effects. The red, green, and blue channels should remain zero until the alpha channel is saturated, as shown in Figure 5.9.1.

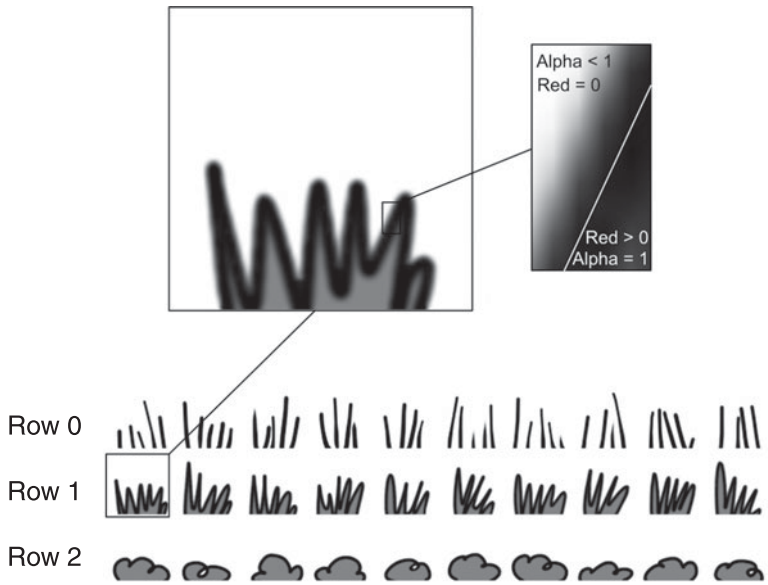

FIGURE 5.9.1 The texture atlas uses red as a color key inside of the graftal imposter and the alpha channel to smoothly blend the graftal imposter with the rest of the scene.

Once you reach the middle of the outline, it blends smoothly into solid red, leaving the alpha channel saturated. The inside of graftal imposters will be blended with the color of the underlying geometry while the outside will be blended with the rest of the scene.

Control Texture

The control texture enables the designer to specify where a graftal imposter may be placed and the type of graftal imposter to be drawn. The alpha channel is used to indicate areas where no graftal imposters can be drawn. Red should be used where the designer wants graftal imposters from the first row of the texture atlas, green for the second row, and blue for the third and final row. Using three color channels isn't the optimal encoding; however, it simplifies the asset-creation process.

Vector Field

It is often desirable to indicate a direction, or flow of the graftal imposters. One example of this comes when using this technique to create fur (or hair) on a character. You could use the normal as an extrusion direction; however, this limits the amount of control the end user has over the final look of the scene. Hair, plants, trees, and so on, don't always grow at a right angle to the surface from which they protrude. To solve this, you can create the vector field, which gives you the direction. See Figure 5.9.3.

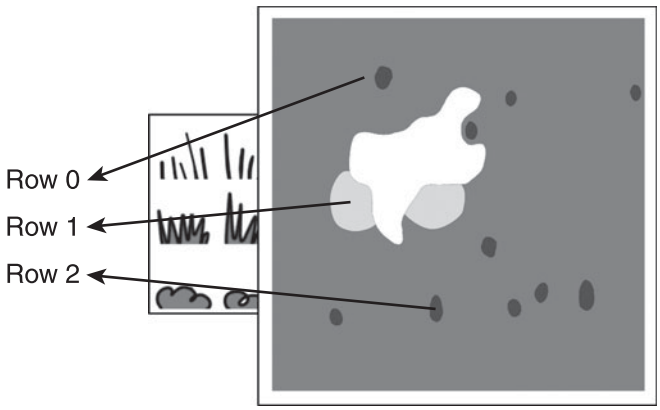

FIGURE 5.9.2 The control texture indicates coverage and row selection.

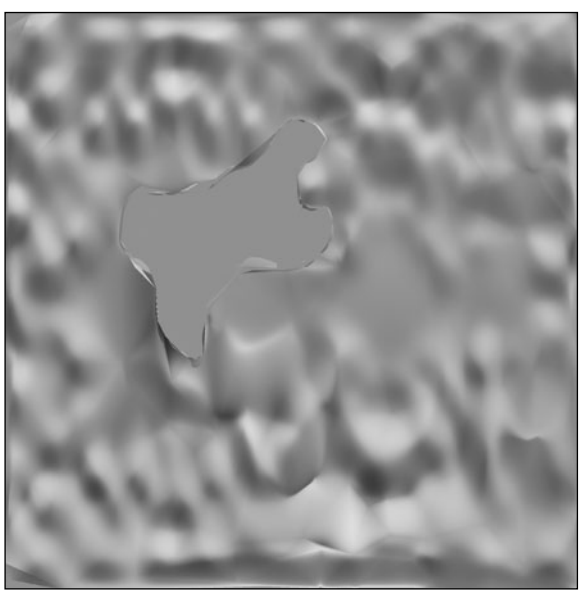

FIGURE 5.9.3 The vector field texture indicates the direction of the graftal imposters.

To create the vector field, you leverage existing digital content-creation applications and plug-ins. After creating a mesh of the desired resolution, the designer saves it and substantially reduces the tessellation of the mesh. You want to be sure to preserve the original normals, as they are used in another step and passed with the mesh. Next, the designer manipulates the normals to indicate which direction the graftal imposters should go; this can be done on a per-face basis or by selecting several normals at the

same time. Once the manipulation is complete, these new values can be saved using a normal map plug-in creating the vector field. The result will look like Figure 5.9.3.

Color Texture and Mesh

The color texture is used to indicate the color of the mesh as well as the internal coloring of the graftal imposters. Figure 5.9.4 is the color texture for the scene. This technique creates the graftal imposters along the edges. Large differences in edge length result in visible irregularities in the width of the graftal imposters; therefore, the mesh should contain triangles of roughly uniform size. In areas with a high level of detail where extremely small triangles are required, it might be best to use the control texture to omit the creation of graftal imposters.

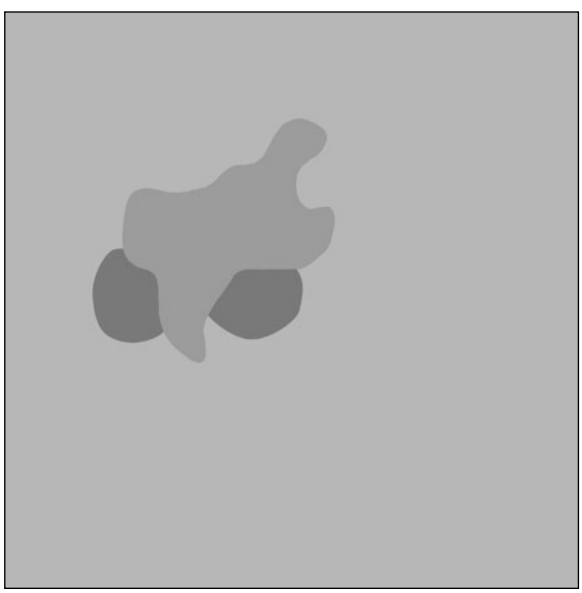

FIGURE 5.9.4 The color texture is used to color the base mesh and the graftal imposters.

Runtime

You can now use the assets created in the previous step during the runtime component of the algorithm. This implementation of graftal imposters requires the use of a programmable graphics language with a geometry shader. First, you draw the original mesh and apply the color texture. Next, you use the geometry shader to determine where to place the graftal imposters as well as which type of graftal imposter to place. Finally, you use the pixel shader to give the final color to the graftal imposters and blend them with the rest of the scene.

The control texture was created earlier to dictate where you can create graftal imposters as well as what type of graftal imposter to draw in a given area. You need to test the triangle to determine whether the primitive is eligible for a graftal imposter and assign a type if it is.

```
texCoordCentroid = ( vertex1uv + vertex2uv + vertex3uv ) / 3;
controlSample = controlTexture.sample( sampler, texCoordCentroid );
if( controlSample.a == 1 )
   if( controlSample.r == 1)
      glyphType = 0;
   elseif( controlSample.g == 1)
      glyphType = 1;
   else
      glyphType = 2;
```

Sampling a texture from within the geometry shader is allowed using the unified instruction set provided with Direct3D 10. You need to choose a point at which to sample the texture, since you have access to multiple vertices. The previous pseudocode shows how you can sample the control texture using the centroid of the triangle currently being processed.

Now that you know the triangle is eligible for graftal imposter(s), you need to test each edge to see whether it is a silhouette edge. A silhouette edge is an edge that's shared by both a front and a back facing triangle. In order to test an edge to see whether it is a silhouette edge, calculate the dot product of the face normal $\mathbf{N_1}$,$\mathbf{N_2}$ with the view direction $\mathbf{V}$ for both faces and test to see if the signs differ [Lake00].

$$(\mathbf{N_1} \bullet \mathbf{V}) * (\mathbf{N_2} \bullet \mathbf{V}) \leq 0 \tag{5.9.1}$$

To create the new geometry at the silhouette edge, you extrude vertices $\mathbf{V}_0$ and $\mathbf{V}_1$ in a direction $\mathbf{D}$ obtained from the vector field sampled using the texture coordinates at the midpoint M of the edge. See Figure 5.9.5.

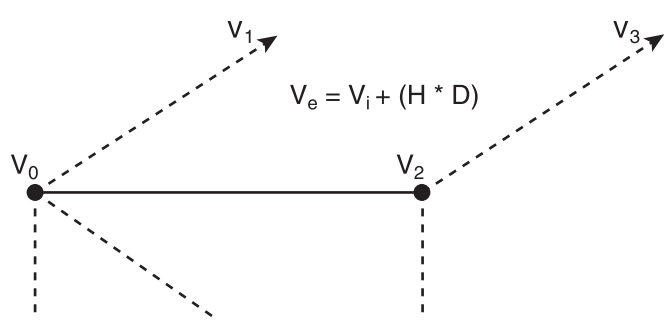

FIGURE 5.9.5 New vertices are created by extruding each vertex along the edge where graftal imposters are desired.

```
//Recreate the original vertex V0
Position = Input.Position;
Output.Position = mul(Position, ObjectToProjection);
Output.GraftalImposterColor = V0Color = ColorTexture.sample
                              (sampler, Input.V0.Texcoord);
. . .
AppendVertex();

//Create a new vertex in the appropriate direction
Position = Input.Position * Direction + GraftalHeight;
Output.Position = mul(Position, ObjectToProjection);
Output.GraftalImposterColor = V0Color;
. . .
AppendVertex();

//Recreate the original vertex V1
Position = Input.Position;
Output.Position = mul(Position, ObjectToProjection);
Output.GraftalImposterColor = V1Color = ColorTexture.sample
                              (sampler, Input.V1.Texcoord);
. . .
AppendVertex();

//Create final new vertex, finishing the quad
Position = Input.Position * Direction + GraftalHeight;
Output.Position = mul(Position, ObjectToProjection);
Output.GraftalImposterColor = V1Color;
. . .
AppendVertex();
```

This pseudocode shows how to create the graftal imposter surface as well as sampling the color texture in order to shade the graftal imposter. Selecting the color once per incoming vertex allows you to have a graftal imposter that crosses a color boundary, because the value is interpolated as it is passed to the pixel shader.

Next, you assign texture coordinates to the newly created geometry to place the graftal imposter on the newly created surface by indexing into the texture atlas. You use the graftal imposter type G to index into the correct row of the texture and a pseudorandom value such as a sample into a noise texture to determine the column C. N_C is the number of variations, or columns, the texture atlas contains (see Equations 5.9.2–5.9.5).

$$uv_{V_0} = \frac{C}{N_c}, \frac{G}{3} + \frac{1}{3} \tag{5.9.2}$$

$$uv_{V_{0New}} = \frac{C}{N_c}, \frac{G}{3} \tag{5.9.3}$$

$$uv_{V_1} = \frac{C}{N_c} + \frac{1}{N_c}, \frac{G}{3} + \frac{1}{3} \tag{5.9.4}$$

$$uv_{V_{1New}} = \frac{C}{N_c} + \frac{1}{N_c}, \frac{G}{3} \tag{5.9.5}$$

The final step is to sample the texture atlas using the texture coordinates calculated in the geometry shader. You want to have a smooth transition from the graftal imposter's black outline to the internal color passed in by the vertex shader. To accomplish this, the red channel of the result is masked off and the sample is linearly interpolated with the color value passed in via the geometry shader. The red channel is used as the blending factor in the interpolation.

```
//coloring the graftal imposter
AtlasColor = AtlasTexture.Sample(Sampler,
             GraftalImposterTextureCoords);
RedZero = float3(0,AtlasColor.gb);
GraftalImposterColor = lerp(RedZero.rgb, IncomingColor.rgb,
                       AtlasColor.rrr);
GraftalImposterColor.a = AtlasColor.a;
```

The incoming color is interpolated across the two vertices that you sampled within the geometry shader. You blend the graftal imposter's soft edge by doing a linear interpolation with the red channel masked out. Preserving the alpha value enables you to blend the outside of the smooth edge with the underlying landscape color. See Color Plate 12 for a full-color example of rendering with graftal imposters.

Acknowledgements

The author would like to thank Jeffery A. Williams, Rahul Sathe, David Bookout, Nico Galoppo, Adam Lake, and the Advanced Visual Computing team at Intel for their assistance, support, and contributions.

Conclusion and Future Work

This gem has shown how to create a scene in a style similar to that found in Dr. Seuss's children's books. The technique utilizes the new technology capabilities of geometry shaders in this GPU-centric technique, leaving more CPU cycles for game logic and other tasks. Currently, we are applying graftal imposters only to the silhouette edges. In our future work, we would like to automatically generate the vector field used for extrusion directions without an artist having to encode it for the entire geometry.

When implementing this technique for production, a couple of additional features may be desired. Adapting the introduction and removal of graftal imposters to provide for inter-frame coherence by scaling or "fading" in the graftal imposters is one possible solution. Another important consideration is handling of z-fighting.

References

[Doss07] Doss, Joshua A. "Inking the Cube: Edge Detection with Direct3D 10," *Game Developer Magazine,* June/July 2007, pp. 13–18.

[Kowalski98] Kowalski, Michael A., et. al. "Art-Based Rendering of Fur, Grass, and Trees," Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1998, pp. 433–438.

[Lake00] Lake, Adam, et. al. "Stylized Rendering Techniques for Real-Time 3D Animation and Rendering," Proceedings of the 1st International Symposium on Non-photorealistic Animation and Rendering, NPAR, 2000, pp. 13–20.

[Rost06] Rost, Randi J. *OpenGL Shading Language,* Addison Wesley, 2006.

[Seuss71] Seuss, Dr. *The Lorax,* Random House, Inc., 1971.

[Smith84] Smith, Alvy Ray. "Plants, Fractals, and Formal Languages," *Computer Graphics,* Vol. 18, No. 3, SIGGRAPH 1984, pp. 1–10.