Interaction with graftals

Pieter Noordhuis
Supervised by dr. T. Isenberg and prof. dr. M. Aiello

January 15, 2009

1 Introduction

An artist can easily draw a few strokes on a piece of paper and create an image of a grassy field or a tuft of grass. While such a drawing may seem to look natural to the human eye, any standard computer rendering of a grassy field would lack this natural look. This lack of expressive power of standard computer rendering is tried to tackle by using graftals in the way specified in this article.

Graftals allow designers to stylize models in a 3D scene. They are usually used for representation of natural structures such as trees, leaves, grass and fur.

In related work, graftals were specified at compile-time and did not provide the designer with much versatility. Interaction with graftals was little possible, which in combination with modern graphics hardware is tried to improve in this work.

2 Related work

In the article by Markosian et. al. [1], a first attempt was made to create a framework where users could interact with graftals. As this was the main basis for this work, a summary will be given of the concepts described in this article.

The lead up to this article was the need for a way for designers to mimic strokes drawn by an artist in 3D space. Because was not possible up until than, the theory put forward in this article was rather new. Limited by the processing power of computers in the time the article was written, any real-time interaction with the graftals or the scene was only possible in a limited way.

Graftals were defined as procedural textures that can be assigned to 3D primitives and behave according to several parameters such as camera position, camera direction and distance to the viewpoint. Every 3D primitive (such as a sphere) can have one graftal assigned, which means that this

graftal will draw itself all over this primitive. In this article, 3D primitives to which graftals can be assigned are referred to as *patches*.

An important part of the specified framework is the way it handles locating primitives in an arbitrary 3D scene. This is dealt with using an *ID reference image*. This image is an offscreen rendering with lighting and blending disabled, that depicts every visible face with an unique color. Using this technique, it is possible match every pixel to its corresponding face in $O(1)$ time. Naturally, this image needs to be re-rendered on every change in camera position or direction.

For each graftal, an off-screen rendering called an *color reference image* is made. This image contains a rendering of all patches that have this graftal assigned to it. Darker tones in this image correspond to a higher density of graftals, where lighter tones correspond to a lower density. This density distribution is used to fulfill the aesthetic requirement of graftals being placed more dense around the silhouettes of patches and less dense in the center. Of course, other distributions of this density can be used to generate other effects.

The tone of a pixel in the color reference image is called the *desire* that a texture is drawn there. Once a texture has been drawn on that pixel, a blurred version of that texture (which size is mostly larger than one pixel) is subtracted. To decide where to draw a graftal, the color reference image is checked for the pixel or pixels with the largest desire that a graftal is placed there.

After the graftal is drawn at the pixel with the largest desire, and the color reference image is updated with this newly drawn graftal, the algorithm repeats these steps until all pixels have a desire value lower than a certain bound.

To decide how to draw a graftal, calculations are made with respect to object space size, screen space size and camera orientation to make decisions on how to scale and orient the graftal. For the graftal orientation, the ID reference image is checked for the face the graftal is drawn on. The surface normal on that position of the face is used to orient the graftal texture before it is drawn.

This drawing of graftals is done per graftal in a separate buffer and later overlaid onto the scene rendering. This process repeats itself for every frame, relying on the previous frame for frame-to-frame coherency reasons.

The specification of graftals is done using a single poly-line and a table of widths (as seen in figure 1). This definition is all but natural and requires technical insight of the internal structures used by the algorithm to be specified. As this requirement does not apply to all designers, this specification does not allow designers any expressiveness.

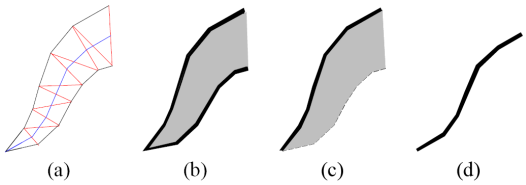

Figure 1: An example graftal by Markosian et. al. [1]. (a) shows the poly-line with the widths projected on it. (b) and (c) show example rendering with strokes around the edges. (d) shows the stroked rendering of the poly-line only

3 Concept

In previously discussed article [1], the notion of *graftals* is based on stoke-based textures, that vary in size, stroke-width and level of detail according to camera parameters. Their configuration is done at compile-time and are therefore not easily modified. Their placement is done per-patch without any density function and therefore put forward a set of restrictions to the user.

In this work, graftals are seen as plain textures instead of a poly-line, with the parameter dependency as used in Markosian [1]. Graftal textures are projected on a local frame (or surface), where this frame itself is subject to the parameters. This frame is defined by the point on the model where it is placed, the surface normal on that point and any graftal specific parameters.

By not restricting the designer on the structure of a graftal and simply letting him define a texture, it is expected that more detail can be packed into graftals and designer can define graftals more freely.

4 Implementation

This section will discuss the implementation of algorithms used by the application made for this assignment.
In contrary to the methods described in related work, this approach uses OpenGL to render the graftals. As computer hardware has been getting faster over the years, it is now possible to rely on hardware rendering of the graftals, instead of drawing them in a buffer yourself.

Because this requires an inherently different algorithm than the one using a color reference image, the algorithm will be described here.

The described algorithm uses an ID reference image to let pixels in renderings correspond with faces in the 3D scene. This technique is also used in this implementation, but only for knowing *where* to place graftals, not as reference when drawing the graftal textures.

In contrary to the placement algorithm described in [1], this implemen-

tation provides possibility to place graftals only single faces, or distribute them on a mesh. Once a graftal has been placed, is it rendered together with the scene, by the OpenGL renderer.

To generate the same effect as in the related work of graftals always facing the camera, graftal textures are placed in a local frame, which rotates around its normal towards the camera. This local frame (as defined by Markosian in [2]) is implemented as an OpenGL quad. This frame depends on 3 different parameters, being: point p in object-space where it is placed, the surface normal $\vec{n}$ at that point and the current camera viewing direction $\vec{v}$. Relying on these 3 parameters, a frame is calculated which will hold the texture. This is done by calculating the cross product $\vec{x} = \vec{n} \times \vec{v}$. This cross product $\vec{x}$ is perpendicular to the surface normal $\vec{n}$, so it can be used to compute a frame at point p. Another parameter we need for this frame computation is a scaling factor s, by which to scale the frame. As this parameter depends on the object-space size of the mesh it is placed on, this parameter is defined per graftal.

Using the vectors $\vec{n}$ and $\vec{x}$, it is possible to compute the four points in object space defining the local frame, being:

$$p_1 = p + s \cdot \vec{x}$$

$$p_2 = p - s \cdot \vec{x}$$

$$p_3 = p_1 + s \cdot \vec{n}$$

$$p_4 = p_2 + s \cdot \vec{n}$$

These four points are used to create an OpenGL quad. This quad gets assigned the texture defined by the graftal and is than rendered by OpenGL just as all the other primitives.

The quads are now subject to the camera orientation to calculate visibility and size. To generate the effect that graftals are seen mostly around the edges, alpha blending is used. Alpha blending fades graftals away when they get drawn oriented more towards the camera position. To calculate a blending factor, the dot product $\lambda = \vec{n} \cdot \vec{v}$ is used with $\vec{n}$ the surface normal and $\vec{v}$ the camera direction. One could choose alpha value $\alpha = 1 - \lambda$ to make sure graftals positioned at the edges of meshes are fully visible, whereas graftals positioned more to the center of objects become less visible.

Because OpenGL quads are used to render the graftal textures, we need to sort them in a certain way. The OpenGL renderer automatically culls objects placed behind other objects. As we may place graftal behind other graftals, and they may include transparency, the graftals need to be drawn starting with the ones the farthest away. As the camera moves, we need to resort all the graftals for every frame. This is done by using the quick-sort algorithm on the values acquired by computing the distance of the camera

to the graftal. Because the actual distance is irrelevant, the square root operation is left out to make this operation faster.

5 Application

The previously described algorithm is implemented in an accompanying application. The process of making this application will be discussed in this section.

As stated before, the main challenge was to let users interact with graftals in a natural way, without limiting them in any way. It should be possible to draw graftal textures, configure their parameters and place them in a 3D scene in an integrated environment. A screenshot of an intermediate version of the application can be seen in figure 2.

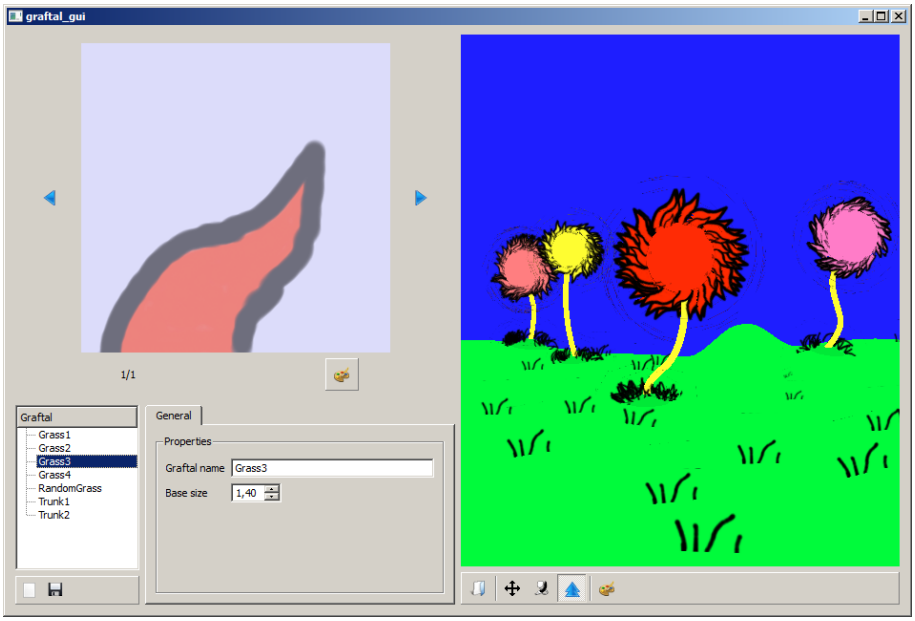

Figure 2: A screenshot of the application showing the graftal drawing pane and library on the left, and the OpenGL renderer on the right. The used scene is modeled after the truffula scene used in Markosian [1].

For loading 3D scenes, the widely used 3DS format is used. This gives users the power of using tools as 3D Studio Max or Blender to model their scenes they want to provide with graftals and load them in the application. This format provides the specification of different meshes, their colors and camera position. Other properties of the 3DS format also provide integrated texturing of meshes and lighting, but are not used within the scope of this application.

The application provides a so-called *graftal library*, in which users primarily can specify graftal textures. Parameters like the previously discussed scale factor and alpha blending can also be specified, but not at first sight. The graftal textures are specified by means of simply drawing them. The application provides a simple though effective panel on which users can draw the graftal textures by means using a mouse or a Wacom tablet. If a Wacom tablet is used, the pen pressure is used to scale the pen-size.

This graftal library stores the different textures per graftal in the PNG[1] format and their configuration parameters in the XML[2] format. This allows users to use other imaging applications (such as Photoshop) when the desired texture style is not feasible using the simple drawing panel, to specify the textures and load them back into the application. The XML format to store the different parameters was chosen because it allows simple modification of the parameters outside the application.

On starting the application, the standard graftal library (located in `graftals/`) will be loaded. On quitting the application, all the currently loaded graftals will be saved. By interchanging different `graftals` directories, users can store different libraries providing different graftals.

The application also provided an OpenGL renderer. Users can load any 3DS scene into the application. The rendering will always take place in real-time, giving the user a natural impression of their interactions. The user is provided with several modes for interaction with the scene. A user can increase or decrease the density of graftals on a certain mesh, by selecting the mode, picking a mesh to work with and dragging the mouse up or down to increase or decrease the density of graftals for that mesh. It is also possible to just draw one graftal on a face, allowing the user to carefully place his graftals on several parts of the mesh. The active graftal in the library will be used when applying these actions.

Camera movement is implemented by using the left mouse-button for directional movement, the right mouse-button for strafing along the plane perpendicular to the viewing direction and the scroll-wheel for zooming in and out.

All camera movement is administered in both OpenGL matrices and custom matrices to make is possible to calculate the camera position and direction in object space at all times.

As there is a graftal library, the application also provides a *scene library*. Users can load any 3DS scene inside or outside of the library and place graftals. This placement is then saved into the scene library, as well as the scene used. This provide a way to save fully decorated scenes for later

[1] Portable Network Graphics
[2] eXtensible Markup Language

reference and/or modification.

This application depends on the following libaries:

- Qt 4.3.4 for the GUI
- Glew 1.5.0 for wrapping OpenGL extensions
- lib3ds 1.3.0 for loading 3DS files
- GLT OpenGL toolkit 1.23 (the math library) for dealing with matrix mathematics

6 Summary

With the article by Markosian [1] as a basis, this new approach provides a solid basis for interacting with graftals at interactive rates made possible by modern hardware. The application lets users create graftals and place them in an arbitrary 3D scene, while allowing them to change parameters in real time. Graftal specification is made more versatile due to the used storage formats and the integrated drawing panel in the application.

7 Future work

While this framework provides a nice basis, more work can be done in several areas. The interaction with the application is possible using a mouse or Wacom tablet, but investigation can be done on using input devices such as multi-touch input screens and the Wiimote.

Rendering of the graftals is now done in a somewhat naive manner, by rendering dense graftal distributions even if the final mesh they are rendered on is only a few pixels of size in the final rendering. As not all graftals could possibly contribute to the final rendering, some way of filtering graftals that should not be rendered should occur to speed up the rendering process.

The sorting of the graftals is another point of improvement. All graftals get sorted on every frame right now, which also poses some performance issues when working with large sets of graftals.

References

[1] Michael A. Kowalski, Lee Markosian, J. D. Northrup, Lubomir Bourdev, Ronen Barzel, Loring S. Holden, and John F. Hughes, *Art-based rendering of fur, grass, and trees*, SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques (New York, NY, USA), ACM Press/Addison-Wesley Publishing Co., 1999, pp. 433–438.

[2] Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Loring S. Holden, J. D. Northrup, and John F. Hughes, *Art-based rendering with continuous levels of detail*, NPAR '00: Proceedings of the 1st international symposium on Non-photorealistic animation and rendering (New York, NY, USA), ACM, 2000, pp. 59–66.