CS224: Interactive Computer Graphics

Art-based Rendering with Graftals

Due: 3/13/10, 11:59 PM

1 Introduction

Between Ray in 123 and Photon Mapping that you just did, you've now had a good deal of exposure to photorealistic rendering. For a change of pace, we're going to do a little non-photorealistic rendering. The goal of this project is to be able to render a model in the style of our dear Dr. Seuss.

You'll be implementing pieces of this paper: "Art-based rendering of fur, grass, and trees", available in docs on the CS224 website.

Read it. This handout contains a brief overview of the algorithm but the devil is in the details... and those are in the paper.

2 Requirements

You will be required to render a mesh model in the style of Dr. Seuss. The paper talks about how to render a "furry image" for an animation, but we'll only require a still image render for this project. You can think of this rendering

as consisting of three major parts

- Rendering References Images
- Finding the 3d location of your graftals using the difference image algorithm.
- Positioning and drawing your graftal into place.

More on each of the parts below.

The paper mentions that there are many different graftals you can use. For this project you will only be required to create the fur graftal for the basic implementation.

This project leaves a lot of room for artistic license and tweaks to make things look better. In addition to implementing the basic requirements, you will also be expected to come up with an addition to your project to make it even nicer. You'll probably get a few ideas for what might look nice after you implement the basic features of the project. You could try to implement the necessary components for animation, instead of just a still frame. You might also want to read the follow up paper: Art-based Rendering with Continuous Levels of Detail. (Actually you'll probably read it for section anyway.) These additions don't have to be top secret, feel free to discuss it with the rest of the class, you don't all have to do unique extensions. Once you have your idea, make sure you run it by a TA first to get it OK-ed. Think of this as practice for your final project, where you will also attempt to extend an existing idea.

3 Algorithm

3.1 Reference Image

You will need to render two "reference images". These are images that you will use later on in the rendering process. The first of these is the desire image. The desire image is a rendering of your model that indicates where you would like to have your graftals (your desire to have a graftal in a certain area). The simplest way to do this is to render an image with your model colored white and with the only light at the camera. This will make the edges of your model dark, indicating that you would like more graphtals on the edge.

The other reference image is an ID image. In this image you will want to render every triangle a different color. This allows you to identify the triangle that a pixel was drawn from by color later in the process. You will use the desire image to find the screen space location of your graftals. You can use the ID image to perform a quick intersection test against a single triangle later when you want to know what world space point the screen space location of your graftal corresponds to.

3.2 Difference Image Algorithm

You will use the difference image algorithm to decide where you would like to place your graftals. The general idea is that you would like more graftals where there is more desire. You will do this by finding the pixel on the desire image with the most desire, and then subtracting a gaussian from the desire around it. Rinse and repeat. For each 2d point found, you will also want it's world space point so that the graftals appear to stick to the model. To do this you will use the ID image to figure out which triangle the screen space point would intersect, and then intersect against that triangle for the world space point. You will also need to use and heuristic to figure out how graftals should be scaled according to distance. You would like to the graftals to get smaller, but you don't want them to be too small because that looks ugly.

3.3 Drawing the Graftal

Once you have the 3d point, you will want to draw the graftal (a GL triangle strip) so that it lines up with the surface normal but faces the camera as best as it can. You will also want to apply any scaling that you computed during the difference image algorithm phase. Depending on how much the graftal faces the camera, you will want to draw only the graftal's spine, the graftal with one edge, or with both edges. This heuristic is just for the fur graftal. You could try coming up with other ways to make different kind of graftals behave.

4 Support Code

You will be using G3D in this project. You can find documentation here: http://g3d.sourceforge.net/manual/apiindex.html Support code can be found in /course/cs224/asgn/drseuss. You will be provided with an interface that lets you draw the graftal you would like to appear on your model. Some examples of how to use G3D can be found in the support, such as triangle drawing code and how to render to an offscreen buffer. Some of the classes you will probably use during this project are RenderDevice, which wraps most OpenGL calls, CoordinateFrame, GCamera, Triangle. You might want to start browsing the docs on these classes to get an idea of how you will use G3D.

There will be a lot of files included with the support code. You don't have to worry about most of them. They are mostly code for loading scenes and meshes. The files you will have to edit are

- Dia.cpp
- GraftalFactory.cpp
- ReferenceImage.cpp

- Graftal.cpp

In GraftalFactory you will only have to write parts of onGraphics3D. TODOs have been left where you need to insert your own code.

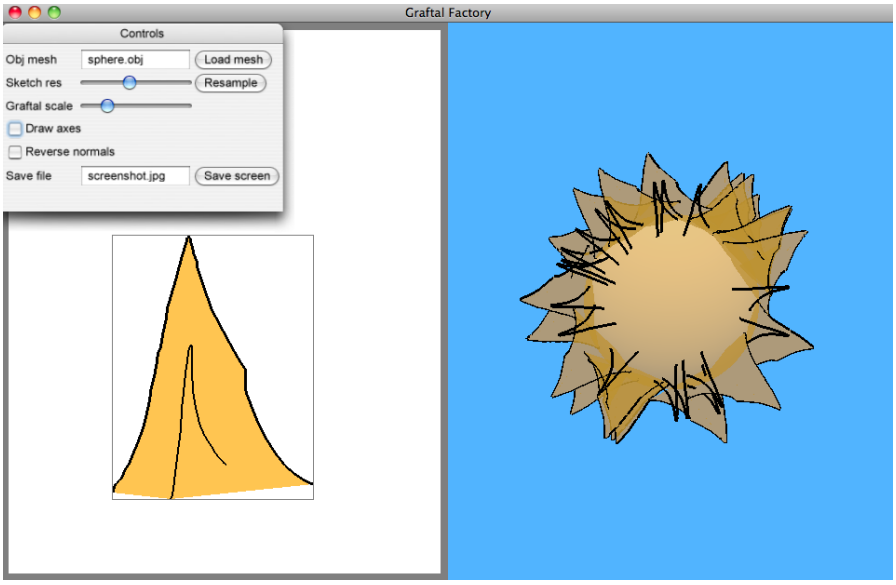
To run your project use : obj/drseuss-d

4.1 SimpleG3D App

We've provided you with a demo G3D app that does some simple rendering and provides you with basic navigational interaction (WASD controls, for those of you who are familiar with first-person shooter video games). In the source code, take note of how certain objects are being placed in the scene (using the CoordinateFrame class) and how their rendering is implemented. We've included some code for rendering triangle strips, which you'll want to pay particular attention to.

4.2 GraftalFactory

In addition to the SimpleG3D App which demonstrates some common functionality you'll need for this project, we've provided you with GraftalFactory, which will be the skeleton for your hand-in. When you build and run this code, you'll see that GraftalFactory provides a split-view display, where the lefthand side is a 2D sketchpad for creating graftals, and the righthand side is 3D viewer for applying and examining graftals applied to 3D models.



To create a graftal: In the sketchpad, left-click and drag the mouse around to create a stroke. When you've created a stroke you like, press the Enter key to commit the stroke. The first stroke you commit will be the spine for your graftal, and the next two will be the upper and lower bounds for your graftal. Upon committing three strokes, your graftal's triangle strip will be created. At this point, the graftal can be applied to your model.

To delete a graftal: With the mouse over the sketchpad, press the Backspace key to delete your graftal and start again.

To resample graftal strokes: By default, a limited number of points from your strokes will be used to triangulate your graftal. If you'd like to increase the sampling, use the slider in the GUI and press the "Resample" button.

To apply your graftal: Graftals may be applied to your model in two ways. For this project, you'll need to implement the Difference Image Algorithm (DIA) to automatically determine 3D graftal positions. Once implemented, running the DIA will be triggered by the "Run DIA" button in the GUI. Alternatively, you may manually include graftals by clicking points on the surface of the model.

To remove graftal applications: To clear your model of graftals, press the Backspace key with your mouse over the display portion of visualization.

To load new models: By default, GraftalFactory loads up a scene with a single elk mesh. We've provided you with some other meshes, but you may also add your own .obj meshes to try out. The meshes are encapsulated within xml scene files that describe where to start the camera in relation to the mesh. Simply type in the path to your xml file and press the "Load mesh" button in the GUI. If you've got some cool meshes, please share them! Also, if for some reason one of your .obj meshes doesn't load properly, please let us know and we'll try to help. Also, some meshes may have different triangle orientations, producing inward-normals. For this reason, we've provided the "Reverse normals" checkbox, which will automatically flip problematic normals. **The xml files and .obj models are located in /course/cs224/asgn/drseuss/models.**

To save images: To save images of your creations, click the "Save screen" button in the GUI. If you've made some cool images, include them in your hand-in!

4.3 Tips for implementation

When first figuring out your difference image algorithm, and especially when trying to use the IDImage to get your 3D point, we suggest you start with just one axis-aligned triangle, get the DIA (difference image algorithm) working on that, then try to get it to work on two triangles. Once you have that , rotate one of the triangles so its not axis aligned, then try overlapping triangles, etc. This will probably save you a lot of pain later on. Also, the paper notes that bucket sort was used to make finding the max desire in the image faster. Don't do this your first time round, do something stupid and foolproof. You can replace it

with more efficient code when everything else around it is working.

To position your graftals in 3D, you may want to first begin by placing spheres at the 3D positions you compute (this will abstract away the issue of orienting your triangle strips). You can see how to draw spheres simply in the SimpleG3D App. When it comes time to orient your strips, you'll need to look at the RenderDevice and CoordinateFrame classes in G3D. Again, check the SimpleG3D App for some pointers here. You may also want to look into RenderDevice::setCullFace, to prevent you from culling backfacing triangles.

5 Demo

To run the demo cd into

```
/course/cs224/demos/drseuss_demo
```

and run

```
obj/drseuss-d
```

6 Handing In

Explain what extension you did in the readme and run:

```
/course/cs224/bin/cs224_handin drseuss
```