ECE 594Q Special Topics in Image Synthesis, Winter 2014

Homework 1 – Suggested workflow

Here are the steps I suggest you follow when doing this assignment in order to complete it successfully:

1. Read the Cook et al. Reyes paper, the RenderMan specification, and the other reading materials posted on the course site. There is a lot here to go through, but try to go through them as much as you can so that you get a better understanding of how RenderMan works.

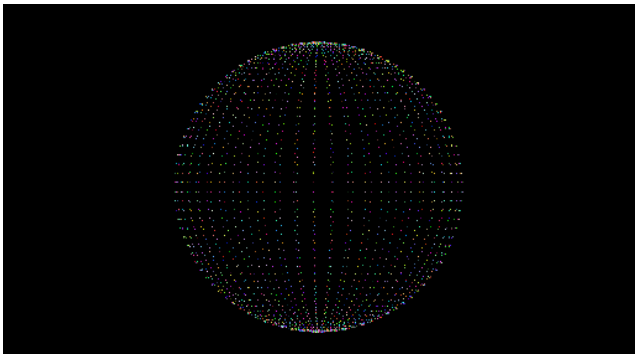
2. Implement the transformation matrices (modelview-projection) to get a point in 3D space to map to the appropriate part of the image. You will want to implement the appropriate rotation, scaling, and translation functions. Create a framebuffer data structure to store the point and write this out to a file. You should now have a program that can take a single point and draw it to the image.

3. Implement a mesh data structure that stores an array of points and their x,y,z positions in space. You will also need to store the surface normal, color, and texture coordinate at each point. Eventually, you will need to set the mesh resolution through a dicing process (in step 8), but for now you should hardcode the size of the mesh so that you can debug this.

4. Pick a single primitive (say a sphere) and implement its appropriate Ri command (in this case RiSphere). This involves taking a mesh using the data structure in step 2 and deforming the mesh to form the shape of the sphere.

   This might sound complicated, but it is actually quite simple. Given an array of points in a grid, can you deform them using cosines and sines so that the points lie on a sphere with specified radius and center? In other words, work out the equations that will take every point on the grid and put it on the appropriate x,y,z position on the surface of the sphere with the specified radius and center.

5. Next, you will need to multiply these points by the modelview projection matrix you worked out in step 2 so that they appear on the screen. You will get a picture that looks like this:



6. At this point, you have a grid of points in screen space. You now have to write the code to do sample the micropolygons of this grid into the framebuffer. To do this, you must first extend the simple framebuffer you implemented in step 2, so that it can take m x n samples per pixel, where these number of samples are given by the command `RiPixelSamples()`. These samples should be on a jittered m x n uniform grid which can be the same for each pixel. To do this, compute a uniform m x n grid and place each sample at the center of each cell and then add a random offset that moves the sample within the grid cell. Make sure you debug this properly.

7. With the samples in place, you should now be ready to sample the micropolygons to produce an image. Loop over each micropolygon and compute the screen samples it overlaps. To do this, set up a bounding box for the micropolygon (simple min/max of the micropolygons vertices in x and y) and test it against each of the samples in the pixels near where it projects to. Remember that you know where the micropolygon roughly projects to because you have the results from computing the modelview transformation.

   For the samples that the micropolygon's bounding box, you need to do an actual test to see if it intersects the micropolygon itself. To do this, divide the micropolygon into 2 triangles and test if the sample intersects either one. Note that this is not 100% correct because of the way the micropolygon can be folded on itself, but it suffices for this class. You can find algorithms to test whether a point lies inside a triangle online. Make sure you debug this thoroughly!

   Once you determine that a sample intersects the micropolygon, for now simply use one of the corner colors (either from the RiColor command or the result of the shading process in step 11). Later, in step 12, you will need to interpolate the sample color from the corners of the micropolygon.

7. Implement the hider using a simple z-buffer. For now, simply check to see if the depth of the new sample is less than the one already stored at that point. If so, replace it with the new one, and store the new closest depth at that sample point. You will later extend this to a linked list to handle opacity (see step X). When you are ready to produce the final image, simply box filter (average) the colors for all the samples in each pixel.

8. Implement the dicing process so that the bounding box of the micropolygon is much smaller than the pixel it projects to.

9. Implement functions for all the other different primitives requested in the homework assignment. Create separate scenes for each primitive, and ensure that the zmin/zmax and thetamax parameters change the appearance of each primitive as defined by the specification.

10. Handle different RiTransformBegin/End pairs for each of the different objects so that you can rotate/translate/scale them in different ways. Since we are not going to handle true nesting of the transformation matrices by keeping a matrix stack, you don't have to worry about that.

11. Compute the intermediate parameters (surface coordinates u, v, derivatives du, dv, surface normal N, etc.) and implement shaders. This basically means that when you want to render an object you first go through and apply a shader function (by following the provided function pointer) for each grid point on the mesh.

12. Improve the sample color quality by interpolating the corner colors of the micropolygon rather than simply using one of them as in step 8.

13. Implement opacity by modifying the framebuffer to store a linked list of objects at every sample instead of simply the nearest value. The objects need to sorted in depth order, with the objects closest to the camera earlier in the list. Remember to clear out the rest of the list when an opaque sample is inserted into the list. Debug this carefully and make sure that you do not have memory leaks, etc. When it comes time to produce the final image, collapse the linked lists at each sample by blending the color samples in the list from back to front order using their given opacities.

14. Implement any remaining commands to get the test scenes provided to render properly.

15. Model some nice test scenes that call some nice shaders

16. Do any remaining extra credit if you have time.

Please pace yourself through this assignment so that you finish within the two week timeline. Do not leave things to the last minute! Also, after you are finished let me know if this workflow was helpful to you or if you have suggestions on how to improve it and make it easier to follow.

Best of luck!