CS 184: Assignment 3 — Real-Time Graphics: OpenGL Scene

Ravi Ramamoorthi

Goals and Motivation

The goal of this assignment is to get a sense for designing interactive 3D computer graphics applications. With the advent of high-end GPUs on every laptop, interactive 3D graphics is now accessible to all, and this ability can be one of the more interesting and fun aspects of the course. We will be using OpenGL, one of the standard graphics APIs (the other common one is DirectX for Windows).

You will create a complete 3D scene, with user controls. The emphasis is on understanding and implementing OpenGL's basic capabilities, as well as using them as building blocks in generating more complex effects. Your main textual reference for this assignment will probably be the OpenGL programming guide. Besides, you may want to look at the material on OpenGL in lecture, and the sample program that was written there. While you can look at the examples there for inspiration, in general, you may not simply copy that code for parts of the assignment. So as not to stifle your creativity, we give you freedom in defining your scene, subject to implementing the functionality specified below. You may also want to take a look at an example of the model assignment, linked from the main assignments web page.

Your project must in general compile and run without the need to link to any libraries or load any dlls, beyond what exists on a standard installation of WinXP with glut and openGL. However, if you feel you absolutely must do something funky and platform-specific, that's probably ok too (check with the TAs and instructor), since we will largely be grading these assignments in demo sessions. You are responsible for finding a machine to demo your program (such as your personal laptop) if it won't run on a standard setup. Including a video of your system in action helps avoid the potential concern of your program simply not running at the demo session.

Your scene must render at interactive rates. You will lose points if it is too slow. In order to focus primarily on the various aspects of the assignment, rather than simply design of textures, we require that *Your entire project, including object files and textures, cannot exceed 10MB* (this does not apply to optional movies showing your system in action).

Finally, I am sure that many of you will want to go overboard with this assignment, and create something at the complexity level of a feature film. However, it makes sense to keep the specific requirements in mind and add features in a modular and incremental fashion instead of having this grand design and having nothing done when the assignment is due. It also makes sense to get the required functionality working before adding optional features.

Due Dates and Logistics

This assignment *must* be done in groups of two. Please speak to the TAs if you are unable to find a partner, so they may suggest combinations. There is simply too much work to be easily done alone.

The due dates are as follows. Partners must be formed at the time of submission of Homework 2 (note however that homework 2 is done individually). The milestone is due on Mar 11. The final assignment is due on Apr 1. You may work on the assignment over spring break if you want (certainly not required), but TA/instructor support may be minimal.

For submission of final assignment, zip up your source code and place it in the submission directory. Then, run "submit as3". Place a separate *README* file that describes how each of the requirements was fulfilled and can be seen by the user. In addition, it would greatly help with grading (but not required) if you include a website (for example, see the model assignment) documenting your work, as well as prepare a video showing your system in action. For the website, providing a link in the *README* is adequate, and you can maintain it yourself, for example on your class account. In this case, please do not modify the website after the due date. If you do these, note as much in the *README*, and include in the directory.

The milestone should be submitted separately, by running “submit as3milestone.” The milestone will be worth about 15 points of the total 100 for the assignment. The goal is to ensure you don’t procrastinate until the last moment, and make the whole assignment a nightmare. Also to alert us to give you additional TA support if you need it (in any case though, if any problems arise at any time, please seek assistance from the instructor or TA).

For the milestone submission, include either a text file (or preferrably simply make it a link to a website you maintain that you don’t modify after the due date). Briefly describe and name your scene, stating what your goals are for the final submission of the project (3 or 4 sentences is enough). Also include at least one image showing the current state of your scene. At this intermediate point, I would expect you at least have most of the objects in there. It is acceptable to not yet have implemented the user interface and animation, and ok if the scene is not yet fully complete (for instance, maybe you haven’t yet got to texture mapping and lighting or implementing the maya and raw file objects). The main point is to provide an intermediate checkpoint to let you and us diagnose problems early. If you can only provide a black screen or don’t quite know how to start, this is clearly a sign of trouble and means we need to work together to make sure you get back on track. In this case, please seek help from the instructor or TA. For the milestone, also consider each of the requirements of the assignment below, and discuss what you have already implemented, and how you plan to fulfil the remaining tasks. Keep this brief; a para or two suffices. The main point is for you to assess honestly where you are and that you have a plan of action to complete the assignment.

Assignment Specifications

The goal is to develop a complete 3D scene. Towards this end, it must be populated with objects (modeling) as follows:

- You must have enough objects for your scene to be considered “complete.” If your scene is a barnyard, you should have a windmill, barnhouse, some animals, trees, and moon or sun.
- One object must be created by hand. You must explicitly type in the data for vertices (normals, texcoords, etc.) between a glBegin() and glEnd(). Do not use any tools to help you with this. It will obviously be a rather simple object, but make it more interesting than a box or rectangle.
- One object must be loaded from a Maya .obj file (read in at runtime). You’ll have to go online and find a unique object that fits well into your scene. The challange here is to scale and place the object well (no giant cats sticking out of the roof of a car.)
- One object must be created from a .raw triangle file (read in at runtime). You will be provided with a tool (see 3dto3d below) to create these files. You must use a for-loop to draw this object. As for the .raw file format, it is the simplest description of geometry possible. It simply lists the vertex locations for all triangles. Each vertex location requires 3 floats, so every 9 floats in the .raw file represents a single triangle. For example, for a triangle with vertices (0,0,0) (0,0,1), (0,1,0), we would simply write 0 0 0 0 0 1 0 1 0.
- You must do all placement, and scaling by hand (or with a code loop). You are not to create a complete scene in some modeling program and then load it all at once.
- At least one object must be textured with an image read from a .tga file (or other image file format of your choice).
- You must texture at least two objects with two different textures. The textures must be placed to fit well on the objects.
- At least one object must be shiny, and one dull in appearance.
- Your scene must have at least one directional light
- Your scene must have at least one point light source.

- You must pick at least one object to be instantiated more than once. You must do this in a way that does not duplicate the entire object, but merely draws it twice with different ModelView matrices. An example was the drawing of the 4 pillars in the program written in class.
- You must be able to switch between fill and wireframe rendering for exactly one object.
- At least half of the objects in your scene (not counting duplicates) must have correct normals so they interact with the lights correctly.
- Use double buffering. hidden surface elimination, and a perspective projection.
- *You must have some mechanism (a key press is most convenient) to toggle on and off all lights and textures. The main purpose of this requirement is for debugging for you, and for us to see that you have created a nice scene using OpenGL even without textures (that is, it's not just slapping down complex textures, but you understand the concepts more).*

In addition to objects and scene modeling, you should have capability for user interaction and animation,

- You must use the mouse and keyboard to allow the viewer (camera) to move about your scene. It is up to you which functions are performed by each input device, but you must use both for substantial user input. Try to make it smooth and intuitive (at least to you).
- The user must be able to look around their current position (pivot around the eye point) The user must be able to move forward and backwards. You may want to implement other appropriate translations If you wish, you may also implement the movement of homework 1, though that is not necessary.
- One of your objects must be constantly animated. In the barnyard scene this would probably be the windmill turning, or an animal walking in circles The user must be able to start and stop the animation of some object. In the barnyard sene, this might be the opening and closing of the barn doors.

Helpful Resources

The class website includes a link to some helpful resources on OpenGL, linked off the assignments webpage.

There is a helper scene that essentially implements homework 1, but with an object that is loaded from a maya .obj file, and a texture. You can use this as a reference, but keep in mind that the design of this code is not clean, and will not scale well. Some students found this code hard to read and understand; it is not required that you use this material at all. If you do look at it, note that the normals are not read properly in the helper code. If you are interested, you can modify the helper code to read this information from the normaldata array in the demo program. Note that if you are developing in unix, you may also find that the endian-ness of your system must be adjusted for.

Apart from this, there is no framework for this assignment; you are completely on your own. If you are having difficulty figuring out where to start, or how to implement some of the most basic functionality, look at the helper scene, the program written in class, or any of the programs in the OpenGL guide.

The tool 3dto3d can be used to convert from one object representation to another. To convert from a 3d Studio Max file, cat.3ds to a Maya obj file, cat.obj, you would type: 3dto3d cat.3ds /if1 /of19 . You can use /of18 for your own instruction, but do not use any such generated code in your scene. Also keep in mind that the normals and texture coordinates are not always preserved when converting between formats.

The tool graphman (or convert in unix) can be used to convert from .jpg to .tga. When you do so, make sure to type in the correct image size (e.g. 128x128), as this tool tends to get that wrong.

In terms of the assignment specification, it makes clear that you should not use any non-standard libraries. Some flexibility will be provided in terms of code to load image formats (for example, you can use textures not in .tga format if you absolutely must), as well as geometric models (if you absolutely need to, you can use something other than the helper for loading Maya obj files. You can also modify the helper code as you see fit).

Extra Credit: Optional add-ons

We describe several optional features you may want for your scene. Please note that while we will give extra credit for these features, the most points will be for implementing the compulsory functionality above, so focus on that first. Relative to the requirements, the number of points per unit effort for the extra credit below will be low.

- *Shading Controls:* Implement controls that allow the user to move between various shading styles like flat, Goraud, Phong, wireframe etc. This might be a useful tool in debugging anyways.
- *Programmable Shaders:* The assignment does not require the use of programmable shaders, which are a common feature on current GPUs. But using shaders can greatly simplify a number of the tasks below. Use shaders to do something interesting.
- *Reflections and Refractions:* Add surfaces like mirrors as well as transparent surfaces (like making a barn wall transparent). Note that OpenGL can't easily do out of order transparency, so you will need to manually specify the order of shading objects.
- *Shadows:* Add shadows on at least one surface. See the OpenGL programming guide for hints and the required transformations.
- *Environment Maps:* Add the capability to have lighting from a complex environment. The OpenGL guide describes how to render perfectly reflective objects. You can also prefilter environments to render diffuse objects. Ask me for details if you plan to pursue this.
- *Advanced Image-Based Techniques:* Images can be used in a variety of ways besides simple texture or environment mapping. The Real time rendering book has more details. One can also use stuff like sprites, that is, 2D images used as impostors for 3D objects and a variety of other approaches. If you're really into it, you could build a renderer based on image warping.
- *Full Scene Antialiasing:* You'll need to render the scene multiple times using the accumulation buffer. See the OpenGL guide for details.
- *Multipass Rendering:* OpenGL provides the basic building blocks on which a variety of realistic rendering effects can be based. Modern video games often use a number of passes, each drawing the same scene with varying parameters, for each frame to create realistic effects. Among the effects possible are soft shadows, full scene antialiasing, motion blur, depth of field, bump mapping, reflections, refractions, compositing, etc. The OpenGL book and the web are good sources of information. Some of these could be simplified with modern programmable shaders.
- *Global Illumination:* You can employ a small amount of particle tracing along with OpenGL to add some global illumination effects like diffuse interreflection, or more complex reflections and refractions. See me if you're interested.
- *Animated Textures:* Allow textures to be time varying on some objects such as a slide show.
- *Parametric or curved Surfaces:* Add curved surfaces other than spheres, cylinders, cones etc. You can also experiment with NURBS surfaces in OpenGL.
- *Hierarchical Scene Graph and Object Instancing:* You may maintain your objects and subobjects in a tree or hierarchical scene graph. This also allows for efficiently instancing objects, hierarchical transforms etc. For the instancing, you should consider nontrivial hierarchically defined objects like a car, not just spheres or cylinders.
- *Level Of Detail:* If your scene has thousands of polygons, you will want to draw only a few of them to keep your frame rate good. You may experiment with keeping versions of your objects at multiple resolutions and using the appropriate one based on the distance of the user from the object, and other metrics. This is particularly useful if your scene includes a terrain model or other complex geometry. Note that given the power of modern graphics cards, you'll need fairly complex models before this has effect.

- *Culling To maintain performance:* 3D applications often avoid drawing unseen geometry. Two useful procedures are view-frustum culling (avoiding drawing objects outside the view frustum) and occlusion culling (avoiding drawing occluded objects). For the former, you could build a bounding box hierarchy of the scene. If a simple test on the upper level bounding volume indicates it doesn't intersect the view frustum, one needn't draw any of the objects. For occlusion culling, one possibility is to precompute visibility relationships, as from certain viewpoints in a room (if you are drawing a palace, maybe you could do a separate computation for each room).
- *Particle Effects:* Include particle effects like steam from a teapot. This is hard to do right; see if you can get something that looks convincing. You could render the particles as small randomly moving triangles.
- *Procedural Modeling:* One may use procedrually computed (perhaps fractal) models for objects like mountainous terrains, plants, fire, smoke etc.
- *Physically based Animation/Collision Detection:* Your animations can be physically based and use notions of dynamics to generate realistic motions. You will want to handle collisions such as a ball bouncing off the ground. Collision detection and handling is quite important in games and can be useful even if you don't have physical simulation built in.
- *Anything else:* Feel free to surprise us with ideas that haven't been mentioned.