CS 184: Assignment 4 — Real-Time Graphics: OpenGL Scene

Ravi Ramamoorthi

Goals and Motivation

So far, you have built structured applications in OpenGL to demonstrate the basics of the OpenGL pipeline. In this assignment, you will build on this knowledge to develop a complete interactive 3D graphics application, of an animated 3D scene. Specifically, 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. While we encourage you to build on the first two assignments and take a modern approach to OpenGL functionality, we will not enforce these restrictions; you are welcome to use standard OpenGL and GLM functions for this assignment.

Your main textual reference will probably be the OpenGL programming guide, as well as the orange GLSL book. 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. Of course, you can use any code from the skeleton or your own work from homeworks 1 and 2, and indeed that would be one of the best places to start. 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, and other examples from the course in previous years.

Given the open-ended nature of this assignment, there are no feedback servers, and we will be grading the assignments largely in one-on-one demo sessions (you will be asked to bring a laptop to demo your assignment; if this is not possible, you should at least have real-time screen capture videos; however, we expect most people to be able to demo their programs running). As such, platform-specific code is allowed, but you are responsible for finding a machine on which the program can be demonstrated (if you think this would be difficult, please coordinate with the GSIs beforehand). We highly recommend you include some real-time screen capture videos, in case your program simply doesn't run 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 recommend that *Your entire project, including object files and textures, not exceed 20MB* (this does not apply to optional movies showing your system in action). You should also implement a feature/toggle to turn textures off.

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.

Logistics and submission

This assignment *must* be done in groups of two. Please speak to the GSIs if you are unable to find a partner, so they may suggest combinations (Piazza also has a new feature for this purpose this year). There is simply too much work to be easily done alone. (If you are comfortable with the workload for this assignment, you are welcome to do HW 5 next by yourself, or if you want to go it alone for any reason. However, this HW 4 must be done in groups of two.)

For submission of the final assignment, zip up your source code and executable, and place it in the submission directory. Then, run "submit hw4". Include a separate *README* file that describes how each of the requirements was fulfilled and can be seen by the user. In addition, it is required to include a simple 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. It is ok for only one partner to submit the assignment, but the other partner should turn in at least a README file saying who they are working with and giving a link to the website. In general, we may not run your code but we would still like to have it in case issues come up, or we want to preserve it as an example for the future. As noted earlier, the main grading will be in person demo sessions where you show your scenes and we go through a checklist of requirements.

Finally, this assignment comes with an earlier milestone. Please note that there are 4 weeks for this assignment, and you need all 4 weeks. The early milestone is to ensure you START EARLY. You will not be able to finish this project if you wait for the last moment. The milestone should be submitted separately, by running "submit hw4milestone." The milestone will be worth about 15% of the grade 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 GSI support if you need it (in any case though, if any problems arise at any time, please seek assistance from the instructor or GSI).

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). The submission requires are file *milestone.txt* 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. Both partners should submit (can be the same information and link to the same website). 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 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 GSIs. 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. At some level, the main difference between this assignment and the previous ones is the existence of a real scene.

- One object must be created by hand. You must explicitly type in the data for vertices (normals, texcoords, etc.) You can see examples of this in the demo program when creating vertex buffer objects. 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 (especially since the demo program already does cubes in this way, so go beyond that, maybe sketch the object out on paper). It is ok to use loops and other similar constructs, but you must have fully designed the object by yourself.

- One object must be loaded from a suitable geometric format such as an .off file or if you want to be fancier (not required), 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 challenge here is to scale and place the object well (no giant cats sticking out of the roof of a car.) You should not use any outside libraries for this purpose (we are aware that reading the full obj format can require a couple thousand lines of code, but you just need to deal with a very simple subset involving object geometry. off files can also be used and are very simple. Similar geometry files are also acceptable. Please contact the GSIs and/or post on Piazza if you have difficulty with this requirement).

- You must do all placement, and scaling "by hand" (or with a code loop) in your program. The main intent is that it is not ok to create a complete scene in some modeling program and then load it all

at once. (Creating one or two objects in a modeling program for fulfilling the previous bullet is ok). Within those constraints, of course your own OpenGL program is not constrained; we just want to make sure you model mainly in OpenGL, and not an external modeler.

- At least two objects must be textured with a suitable image (the images can be in any file format, and can be downloaded from the internet). The GSIs can provide help in terms of code to read in images; in general, you could just use the freeimage library provided, or create a simple ppm file as in the demo program (to/from which you can convert with many utilities). You must use at least two different textures (one texture for each of two distinct objects at minimum). 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 demo program.

- 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 turn off (and on) all 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).*

- You must use programmable fragment shaders with GLSL. Beyond the basic shader you developed in homework 2, you must include at least one interesting lighting effect with the shader, such as shadows, environment maps, bump maps, some non-photorealistic warping and rendering etc. Further effects may be extra credit; see below.

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

- You must use *both* 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. The movement in homeworks 1 and 2 is the baseline, but the viewer also needs to be able to zoom and translate, and use mouse as well as keyboard.

- 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.

In the limit, you can think about designing a *simple 3D video game* (this is of course not required, but used to be a later assignment in this course, and would make a great project for homework 6). For some helpful resources, see the links on OpenGL off the class website.

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 likely be low.

- *Shading Controls:* Implement controls that allow the user to move between various shading styles like flat, Goraud, Phong, wireframe etc. for all objects. This might be a useful tool in debugging anyways.

- *Programmable Shaders:* You can go totally wild with shading effects using the programmable shaders. You can find inspiration and examples in the GLSL book. Obvious ideas are bump maps, displacement maps, etc. but feel free to surprise us with complex shading effects. (Obviously, the first effect will count against regular credit, but you can do much more).

- *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. You should really be able to do shadow mapping for shadows in the entire scene; it's not that hard.

- *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. (example code is also in the glsl book; don't just blindly copy it though).

- *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.

- *Ray Tracing:* For the really ambitious, you could enable some local ray tracing in the shader for things like shadows and reflections.

- *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. A really impressive demo would be to use the instant radiosity technique from SIGGRAPH 97 by Keller.

- *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 (on modern graphics cards, they are so fast this doesn't matter; scale the above sentence up to tens of millions of polygons). 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.

- *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 procedurally 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.