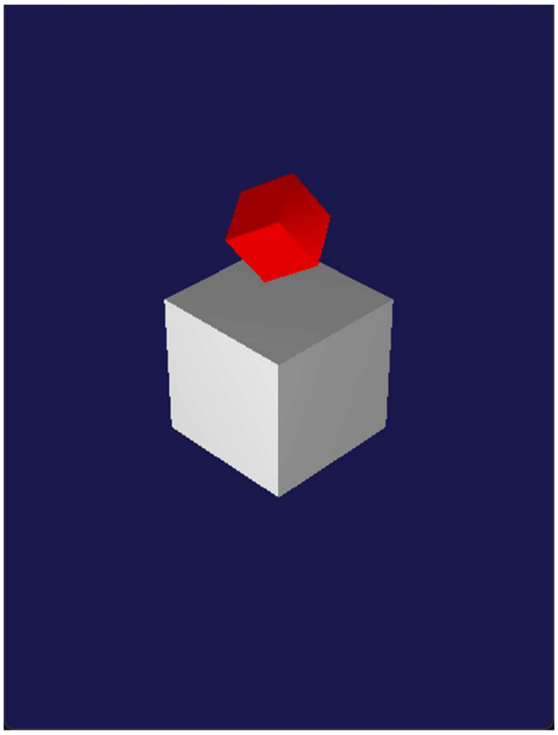
Computer Graphics for Games 2023

Required:
Project Code + Report
(Online submission)

Gem Pedestal

Photorealistic gem on a marble pedestal with
real-time editor like object manipulation.



João Baracho
199248
Computer Science and Engineering
joao.baracho@tecnico.ulisboa.pt

Abstract

My goal for this project was to create a model of a gemstone on top of a marble pedestal, and try to give it a realistic look. I also wanted to have organized scene management, and the ability to manipulate the objects like one would in a 3D editor like blender.

For the scene management I implemented a scene graph to store information about the camera and the lighting of the scene, with scene nodes that would store information about the specific objects (model matrix, mesh, shader and color used).

The scene graph structure also takes care of all manipulation done to the scene, scaling, rotation, translation and object picking, using the mouse to interact and the keyboard to select the interaction mode.

I implemented the Blinn-Phong lighting model (with ambient, diffuse and specular components), but I wasn't able to make the marble texture using procedural noise.

Regarding the scene management and real time manipulation the project went well, but the lighting and the texture were extremely challenging. Paired with poor time management and enough preparation, I didn't have neither the knowledge nor the time to complete the project.

1. Technical Challenges

1.1. Scene Management

Facilitate scene management, creating a scene graph to handle the general scene attributes and functions, with nodes to handle object specific ones. The scene graph would be based on the one presented in class.

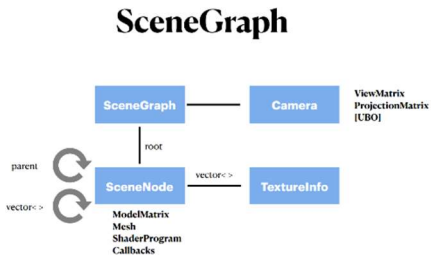


Figure 1: SceneGraph diagram

The scene graph also facilitates the implementation of a way to store and load the whole scene from a file.

1.2. Real Time Object Manipulation

Let the user manipulate the objects in the scene in real time with the mouse and keyboard, specifically scaling (uniform and axis specific), rotation and translation (view plane and axis specific) transformations.

1.3. Scene Lighting

Implement Blinn-Phong lighting to give a realistic look to the scene.

Figure 2: Blinn-Phong Lighting

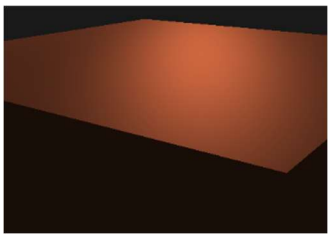


Figure 3: Phong Lighting

1.4. Object Materials

For the gemstone the color with the lighting is enough.

For the pillar, replicate the marble texture using procedural noise to create the veins in the marble.

Figure 4: Marble Texture

2. Technical Solutions

2.1. Scene Management

There are two classes for scene management, Scenegraph and SceneNode.

The Scenegraph class stores aspects of the whole scene: a pointer to the camera and the attributes of the view matrix (eye, center, up) and projection matrix (fovy, aspect, near, far), so we can initialize the camera from the information on the scene graph, the position of the scene light and a vector with the scene nodes. It implements functions to define and initialize the camera, save and load the scene graph and draw it's contents, as well as the callbacks to receive user input from the main app.

The SceneNode class stores information about its specific object: the model matrix separated into the three transformation matrices (scale, rotation, translation), the color, the mesh and the shader used to render the object. It also implements functions to save, load and draw, specific to the object.

The draw function of the scene graph calls the draw function of each of its nodes, which in turn bind the shaders, fill in the uniforms and draw the mesh, specific to that object.

To maintain the code organized and have the meshes and shaders easily accessible across the whole project, there is also a Manager class, a singleton template class that contains only a vector with the saved elements and functions to access them. In this project there's one for meshes and one for shaders, initialized when the application starts. With the managers, instead of having a pointer for the mesh and shader on each node, we have the corresponding ids to access them. This method makes it easier to implement the save and load methods.

To save the scene graph, the relevant attributes are converted to strings and written on a .txt file with their name to facilitate reading (despite requiring one more line of code to read each value to ignore the name). The save function on the scene graph saves the attributes needed for the camera and the light position, and then calls the save function on each node, which saves the model matrix components, the color, and the ids of the mesh and shader.

To load the scene graph, all its initialization functions are called with the parameters read from the file, creating each node the same way.

2.2. Real Time Object Manipulation

The scene graph has also the current manipulation mode, triggered by clicking the corresponding key. Object picking, camera movement and each manipulation can only be active at a time, when the correct mode is active.

The stencil buffer was used to select the object to manipulate. At the start of the scene graph draw method, the stencil buffer is enabled, and the glStencilOp is set to only update the value when it passes the depth and stencil test. At the start of the scene node draw method, the glStencilFunc is updated with the identifier of the current object, and the stencil test is set to always pass, so the stencil buffer is set to only update the value when the depth test passes, which means only when the current object is actually drawn,

so each pixel gets assigned an object. When clicking the scene, the scene graph callback reads the stencil buffer value corresponding to the clicked pixel, getting the object correspondent to that pixel. Every manipulation will be applied to the object with the retrieved identifier.

The objects have three possible transformations:

Scaling: Scrolling the mouse wheel will trigger the corresponding scene graph callback that will call the scale method of the selected node. Depending on the scroll direction, a scale factor is assigned to scale up or down, as a scale vector to be multiplied by the scale component of the model matrix. If **[X]/[Y]/[Z]** are pressed when the scaling is done, only the **x**/**y**/**z** component of the scaling vector will be kept, resulting in scaling only on the corresponding axis (object coordinates).

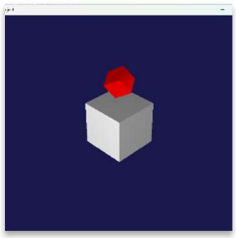


Figure 5: Base model

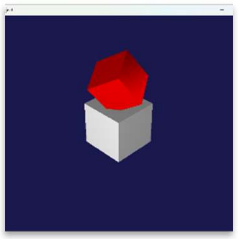


Figure 6: Gem uniform scale

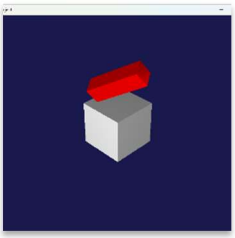


Figure 7: Gem non-uniform scale

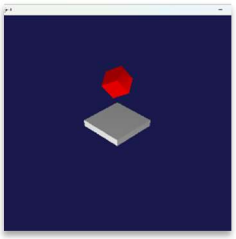


Figure 8: Pedestal non-uniform scale

Rotation: If the left mouse button is clicked, dragging the cursor will trigger the scene graph callback that will call the rotate method of the selected node. The rotation is calculated using two quaternions. One quaternion makes the rotation on the U axis of the eye space, based on the horizontal mouse movement, and the other on the S axis of the eye space, based on the vertical mouse movement. Both quaternions are multiplied by the rotation component of the model matrix.

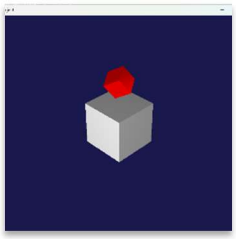


Figure 9: Base model

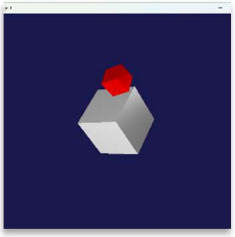


Figure 10: Pedestal rotation

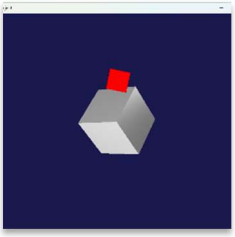


Figure 11: Pedestal and gem rotation

Translation: If the left mouse button is clicked, dragging the cursor will trigger the scene graph callback that will call the translate method of the selected node. The horizontal and vertical distance travelled by the mouse will be multiplied by the normalized S and U vectors of the eye space and then added together to make the final translation vector to be multiplied by the translation component of the model matrix. If **[X]/[Y]/[Z]** are pressed when the translation is done, only the **x**/**y**/**z** component of the translation vector will be kept, making the object move along the corresponding axis (world coordinates).

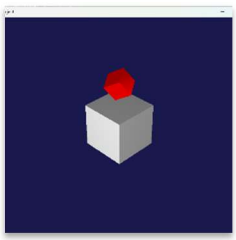


Figure 12: Base model

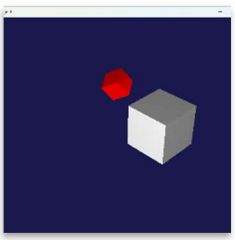


Figure 13: Pedestal left rotation

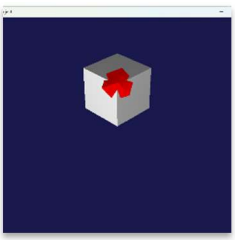


Figure 14: Pedestal upwards translation

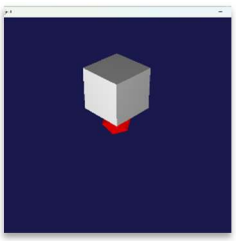


Figure 15: Gem backwards translation

2.3. Scene Lighting

The Blinn-Phong lighting model was implemented through glsl shaders, and it has three components.

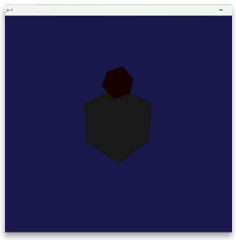


Figure 16: Ambient component

Ambient Component: Multiply the strength of the ambient component (a small constant) with the light color.

Diffuse Component: Send the normal and the fragment position from the vertex shader to the fragment shader. Since we are calculating the lighting in world space, to get the fragment position we need to put the position in world space multiplying it by the model matrix. A similar step is needed for the normal, but with the normal matrix. The diffuse impact is the dot product of the normal and the light direction (light position minus fragment position) vectors (both normalized since we only want their direction), the farther the light direction is from the normal, the less impact, and if the value is negative, it's set to zero. The final value of the diffuse component is the diffuse impact multiplied by the light color.

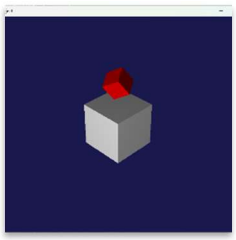


Figure 17: Diffuse component

Specular Component: Add the view direction (eye position minus fragment position) and light direction vectors (and normalize it) to get the halfway vector. The specular impact is equal to the dot product between the normal and halfway vector (more impact the closer they are, set to 0 if negative), to the power of the shininess factor. The final value of the specular component is the diffuse impact multiplied by the diffuse strength (a constant) multiplied by the light color.

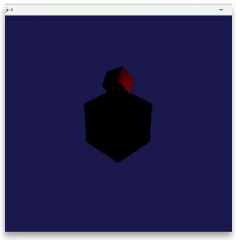


Figure 18: Specular component

These three components are added and then multiplied by the color of the object, resulting in the final fragment color.

2.4. Object Materials

The marble material for the pedestal was not made.

3. Post-Mortem

3.1. What went well?

Using the base code provided by the professor was obviously a good decision given the time limits on the assignment.

Implementing the manager system made the scene management easier, but mainly it made the saving and loading of the scene much simpler, having to save only a string instead of the object.

The save and load methods were much easier to implement that I thought and worked surprisingly well, especially the choice I made to use my own functions to transform de glm vectors and matrices to strings, which made the parsing of the document much easier. Leaving the names of the attributes and what was being saved in the file was also a good choice, because despite requiring one more line of code to ignore the title, it makes it perfectly understandable if you were to read it, which even helped me debugging.

Lastly, the real time manipulation ended up better than I expected as well, especially the translation. Getting the axis from the camera view matrix directly and using them for the transformations made the code really easy and fast to make, and they ended up a great result.

3.2. What did not go so well?

The main difficulty was the awareness of what needed to be done, made worse by poor time management. The first part of scene management, saving and loading and the real time manipulation were quite straightforward, but when I got to the shaders and textures, I realized the material provided (PowerPoints and code examples) wasn't nearly enough, at least for me, to be able to implement them, and at that point it was too late to search and/or ask questions about it, so I ended up using LearnOpenGL website to implement the Blinn-Phong lighting, which ended up not great, and I wasn't able to make the marble texture. This lack of time management made it impossible to find/model the original idea I had in mind for the pedestal and gem.

Also, by using the given base code, I didn't learn how to use and understand OpenGL as well and in depth as I would've liked.

3.3. Lessons learned

Get all trivial tasks out of the way as soon as you can, everything you could do before the course, do it. Attending the classes is not enough, at least for me the examples showed were great to understand the concepts, but one time isn't enough, so try learning about how to use OpenGL and how to do the shaders, textures, etc. outside class.

Finally, as mentioned before (although it's probably not feasible with the limited time), try to implement the whole application yourself so you get practice and understand how to use the OpenGL API.

References

Lecture Slides:

- Viewing Pipeline
- Managing Geometry

Websites:

- https://learnopengl.com