Middle East Technical University | Department of Computer Engineering

CENG 477

Introduction to Computer Graphics

Fall '2021-2022
Assignment 3 - OpenGL with Programmable Shaders

Due date: February 6, 2022, Sunday, 23:55

1 Objectives

The Moon is Earth's only natural satellite. The Moon's orbit around Earth has a sidereal period of 27.3 days. The moon keeps the same face pointing towards the Earth because its rate of spin is tidally locked so that it is synchronized with its rate of revolution (the time needed to complete one orbit). In other words, the moon rotates exactly once every time it circles the Earth. You are going to implement an OpenGL program to render the spherical Earth model and moon orbits it using vertex and fragment shaders to display a texture image as a height map to fly through the scene interactively by processing keyboard input. The input for this assignment is three image files. Namely, the height map and the texture map of the world, and the texture map of the moon. There will be one point light.

Keywords: *OpenGL, programmable shaders, texture mapping, height map, interactive fly-through, Phong shading*

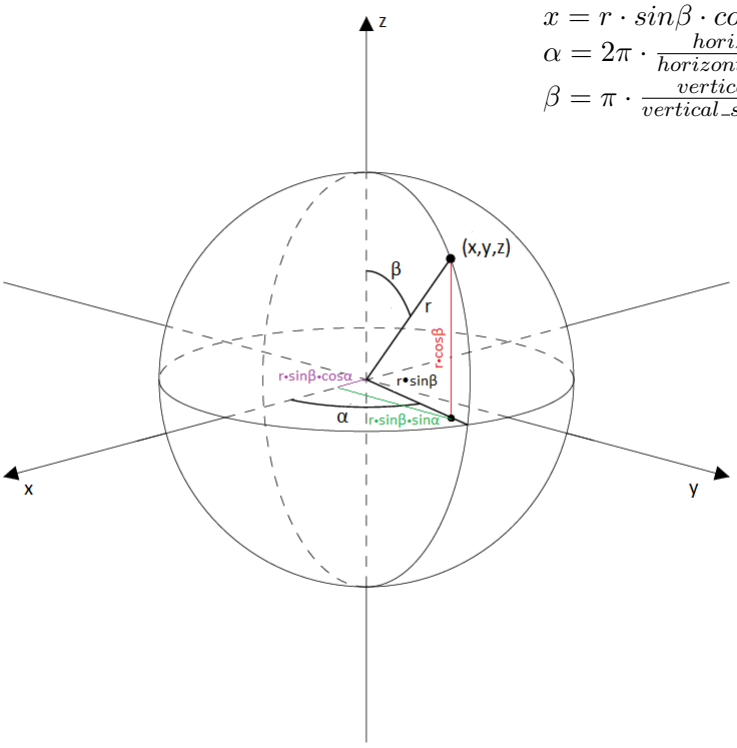
2 Specifications

1. The name of the executable will be “hw3“.

2. Your executable will take three image files in ”jpg” format as a command-line argument. The first file will be used to compute the heights, the last two files are there for texture mapping. One should be able to run your executable with the command;
   >> `./hw3 height_map.jpg earth_texture_map.jpg moon_texture_map.jpg`

3. You will create a sphere where the radius for Earth is 600 units and for the moon is 162 units. You will split them into horizontally 250 pieces and vertically 125 pieces. The center of the Earth will be at (0, 0, 0) and the initial center of moon will be at (0, 2600, 0).

$$z = r \cdot cos\beta$$
$$y = r \cdot sin\beta \cdot sin\alpha$$
$$x = r \cdot sin\beta \cdot cos\alpha$$
$$\alpha = 2\pi \cdot \frac{horizontal\_step}{horizontal\_split\_count}$$
$$\beta = \pi \cdot \frac{vertical\_step}{vertical\_split\_count}$$



4. The keys used for moving or changing the position of elements inside the scene should be handled continuously. Namely, it should not be used with the action of any type; while pressing the key, it should execute the related commands repetitively.

5. The heights, i.e., y-coordinates, of the vertices will be determined in the vertex shader from the corresponding texture color(only R channel) on the heightmap image. The computed height will be multiplied with a height factor to determine the final height. The direction of the height will be computed with vertex normal. The height factor will be initially 80 and the user will be able to increase or decrease this factor by 10 with the R and F keys.

6. The Earth and moon will rotate around their own axis. It will be rotated by $0.5/horizontalSplitCount$ in main render loop.

7. The Earth's origin is placed at (0, 0, 0) and the moon will orbit the Earth 0.02 degree/iteration with the distance of 2600 unit only on x and y axis. Initial orbit degree will be 0. The z axis of the moon will be always 0.

8. There will be one light source in the scene at the initial position of $(0, 4000, 0)$. The intensity of the light source is (1.0, 1.0, 1.0) and there will be no attenuation.

9. You will implement Phong shading for determining the color of surface points. The light calculations will be same for both fragment shaders.

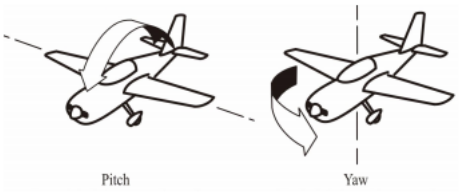
```
Ambient reflectence coefficient = (0.5, 0.5, 0.5, 1.0)
Ambient light color = (0.6, 0.6, 0.6, 1.0)
Specular reflectence coefficient = (1.0, 1.0, 1.0, 1.0)
Specular light color = (1.0, 1.0, 1.0, 1.0)
Specular exponent = 10
Diffuse reflectence coefficient = (texColor.x, texColor.y, texColor↩
    .z, 1.0)
Diffuse light color = (1.0, 1.0, 1.0, 1.0)
```

10. The pixel value retrieved from texture map will have the coordinate of $(\frac{horizontal\_step}{horizontal\_split\_count}, \frac{vertical\_step}{vertical\_split\_count})$.

11. The computed surface color will be the combination of these lights multiplied by the texColor.

12. To implement Phong shading, you will have to compute the normal vectors of the vertices at the vertex shader and define the normal vector as a varying variable so that the normal vectors will be interpolated by the rasterizer and passed to the fragment shader. The normal vector of a vertex is defined as the average of the normal vectors of the triangles that this vertex is adjacent to. You will have to query the heights of the neighboring vertices (hint: use texture look-up for this, too) to compute the normals of the adjacent triangles.

13. You will implement a camera flight mode to be able to fly over the visualized terrain. The camera will have a gaze direction, which will be modeled by two angles for pitch and yaw. See the following figure for the definitions of these terms.



Pitch and yaw angles are very closely related to the phi and theta angles that we have used to represent a sphere with parametric equations. The user will be able to change pitch with W and S keys and change yaw with A and D keys where changing is 0.05 unit. W and S keys rotates gaze around the left vector. A and D keys rotates the gaze around the up vector. Do not forget to update up and left vector with normalizing later. The camera will also move with some speed at every frame. The speed will increase or decrease with 0.01 by pressing the Y and H keys on the keyboard. Initially, the speed will be 0 and the camera gaze will be

(0, -1, -1) and up will be (0, 0, 1). The camera will be positioned initially at (0, 4000, 4000) where w is the width of the texture image. Pressing the X key, the plane will stop. Namely, the speed will be 0.

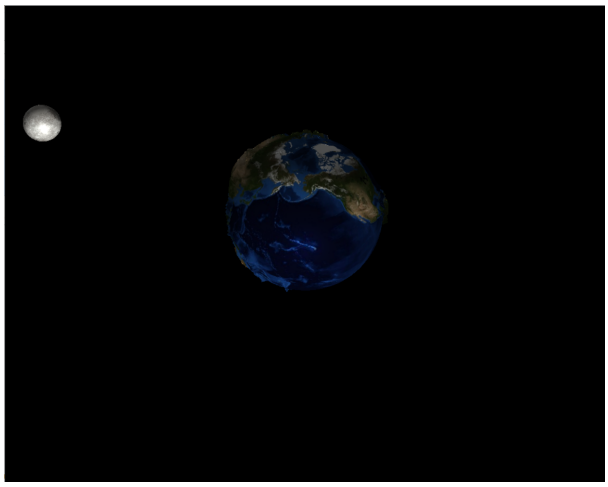
14. When pressing the I key, the plane will be placed to the initial position with initial configurations of the camera and speed of 0.

15. Window size is initially (1000, 1000) and it should be switched to full-screen mode with the key P. Besides, your program should support resizing window operation with the frame.

16. There will be perspective projection with an angle of 45 degrees. The aspect ratio will be 1, near and far plane will be 0.1 and 10000 respectively.

3 Sample input/output

The sample input files are given at OdtuClass. The sample view of the OpenGL display window for the "height_gray_mini.jpg", "normal_earth_mini.jpg" and "moon_mini.jpg" for inputs are shown in the images below.



4 Regulations

1. **Programming Language:** C++

2. **Late Submission:** You can submit your codes up to 3 days late. Each late day will be deducted from the total 7 credits for the semester. However, if you fail to submit even after 3 days, you will get 0 regardless of how many late credits you may have left. If you submit late and still get zero, you cannot claim back your late days.

3. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations and will get 0. You can discuss algorithmic choices, but sharing code between each other or using third party code is strictly forbidden. To prevent cheating in this homework, we also compare your codes with online

sources and previous years' student solutions. In case a match is found, this will also be considered as cheating. Even if you take a "part" of the code from somewhere/somebody else - this is also cheating. Please be aware that there are "very advanced tools" that detect if two codes are similar. So please do not think you can get away with by changing a code obtained from another source.

4. **Newsgroup:** You must follow the discourse (cow.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.

5. **Submission:** Submission will be done via OdtuClass. You can team-up with another student. You will provide a make file to build your homework. Be sure, before submitting, it is running on **Inek** machines. It will not produce any outputs since an interactive OpenGL display window will be used. Create a .zip file that contains your **source files and Makefile**. If your directory structure is wrong or makefile does not work (or you do not have one) and therefore we will have to manually try to compile your code - there will be an automatic **penalty of 10 points** for each. The .zip file should not include any subdirectories. **The .zip file name will be:**

   - If a student works with a partner student:

     `<partner_1_student_id>_<partner_2_student_id>_opengl.zip`

   - If a student works alone:

     `<student_id>_opengl.zip`

   - For example:

     ```
     e1234567_e2345678_opengl.zip
     e1234567_opengl.zip
     ```

   Make sure that when below command is executed, your executable file is ready to use:

   ```
   >_ unzip e1234567_opengl.zip -d e1234567_opengl
   >_ cd e1234567_opengl
   >_ make hw3
   >_ ./hw3 <height_map_file_name> <earth_texture_map_file_name>
   <moon_texture_map_file_name>
   ```

   Therefore you HAVE TO provide a Makefile in your submissions.

6. **Evaluation:** Your codes will be evaluated based on several input files including, but not limited to the test cases given to you as an example. We will evaluate your results visually. Therefore, if you have subtle differences (numerically different but visually imperceivable) when running the program, that will not be a problem. Allowed libraries other than standart libraries for the assignment are only glew, glfw, glm and jpeglib. Using any other library is strictly forbidden. Homeworks implemented with GLUT will be ignored and not be graded. Read the specifications carefully. Anything that is conflicting with specifications will make you lose point/s.