Middle East Technical University | Department of Computer Engineering

CENG 477

Introduction to Computer Graphics

Fall '2019-2020
Assignment 3 - OpenGL with Programmable Shaders

Due date: January 11, 2020, Sunday, 23:55

1 Objectives

The flat Earth model is an archaic conception of Earth's shape as a plane or disk. Many ancient cultures subscribed to a flat Earth cosmography, including Greece until the classical period, the Bronze Age and Iron Age civilizations of the Near East until the Hellenistic period, India until the Gupta period (early centuries AD), and China until the 17th century. Today, there is still a society that tries to prove the concept with some evidence. Since you are already used to spherical Earth model, there is nothing more to say about it. In this assignment, you are going to implement two OpenGL program to render both flat Earth model and spherical Earth model 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 two image files. Namely, the height map and the texture map. There will be one point light.

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

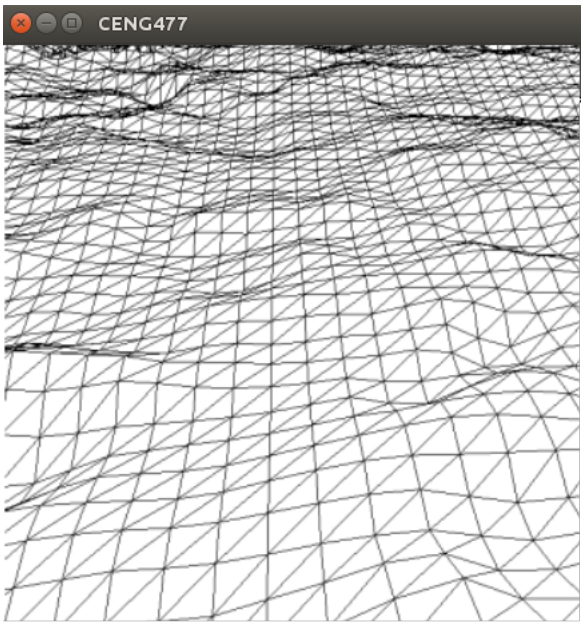
2 Specifications

2.1 Flat Earth(40 pts.)

1. The name of the executable will be "hw3_flat".

2. Your executable will take two image files in "jpg" format as a command-line argument. The first file will be used to compute the heights and the second file is there for texture mapping. One should be able to run your executable with the command;
   >> `./hw3_flat height_map.jpg texture_map.jpg`

3. You will create flat earth in OpenGL with the same resolution of the texture image. Each pixel will be represented by two triangles as in the following image. If the width and height of the texture image is w and h, there will be a total of $2 \cdot w \cdot h$ triangles in your heightmap. The flat heightmap will be on the xz plane with the corner vertices at (0, 0, 0) and (w, 0, h).



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 height factor will be initially 10.0 and the user will be able to increase or decrease this factor by 0.5 with the R and F keys.

6. Q and E keys will move the texture and height map along the plane left and right respectively by subtracting and adding 1.

7. There will be one light source in the scene at the initial position of $(w/2, 100, h/2)$. The intensity of the light source is (1.0, 1.0, 1.0) and there will be no attenuation.

8. The light source can be moved using arrow keys. The height of the light source(y-position) is increased and decreased with T and G keys respectively. The position of the light source is changed by 5 for all directions.

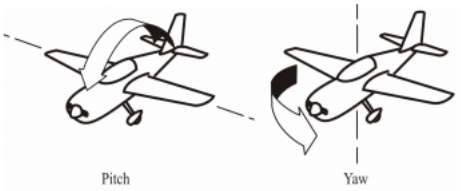
9. You will implement Phong shading for determining the color of surface points.

```
Ambient reflectence coefficient = (0.25, 0.25, 0.25, 1.0)
Ambient light color = (0.3, 0.3, 0.3, 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 = 100
Diffuse reflectence coefficient = (1.0, 1.0, 1.0, 1.0)
Diffuse light color = (1.0, 1.0, 1.0, 1.0)
```

10. The computed surface color will be combined with the texture color do determine the final color of the surface. "clamp" method can be used to combine these two colors.

11. 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.

12. 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, 0, 1). The camera will be positioned initially at (w/2, w/10, -w/4) where w is the width of the texture image. Pressing the X key, the plane will stop. Namely, the speed will be 0.

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

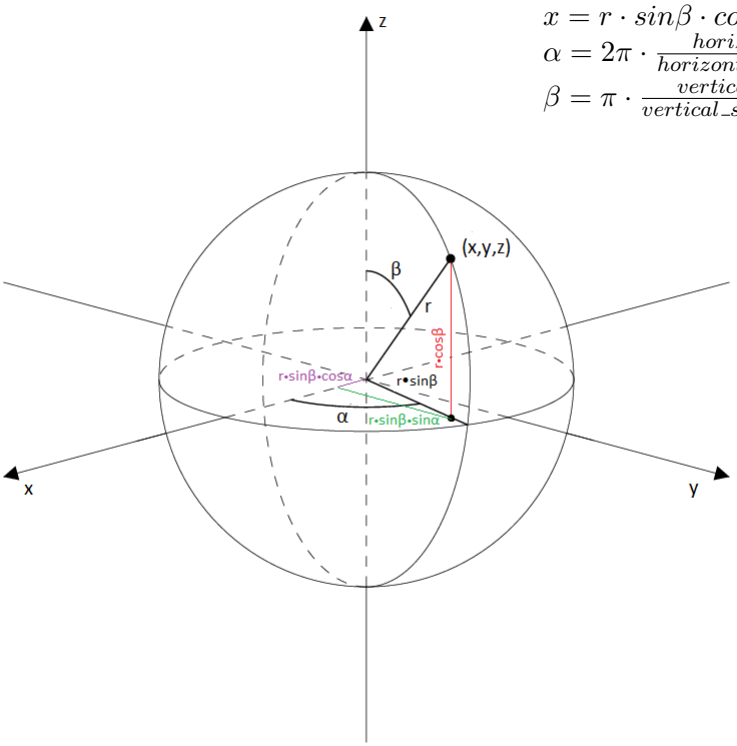
14. Window size is initially (1000, 1000) and it should be switch to full-screen mode with the key P. Besides, your program should support resizing window operation with the frame.

15. 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 1000 respectively.

2.2 Spherical Earth(60 pts.)

1. The name of the executable will be "hw3_sphere".
2. Your executable will take two image files in "jpg" format as a command-line argument. The first file will be used to compute the heights and the second file is there for texture mapping. One should be able to run your executable with the command;
   `>> ./hw3_sphere height_map.jpg texture_map.jpg`
3. You will create a sphere where the radius is 350 units and split it into horizontally 250 pieces and vertically 125 pieces. The center of the sphere will be at (0, 0, 0).

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

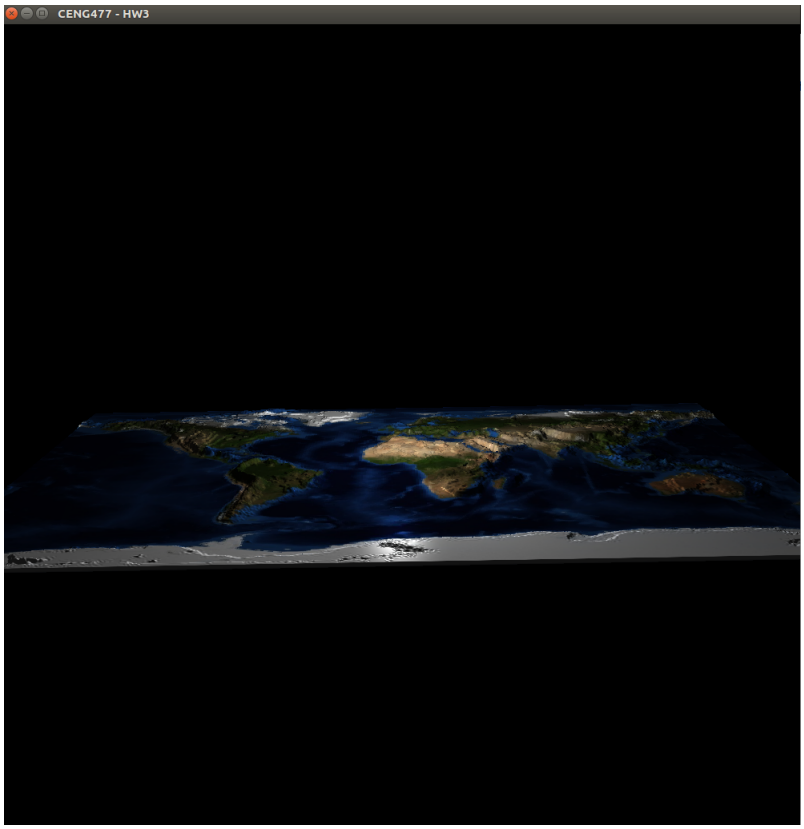


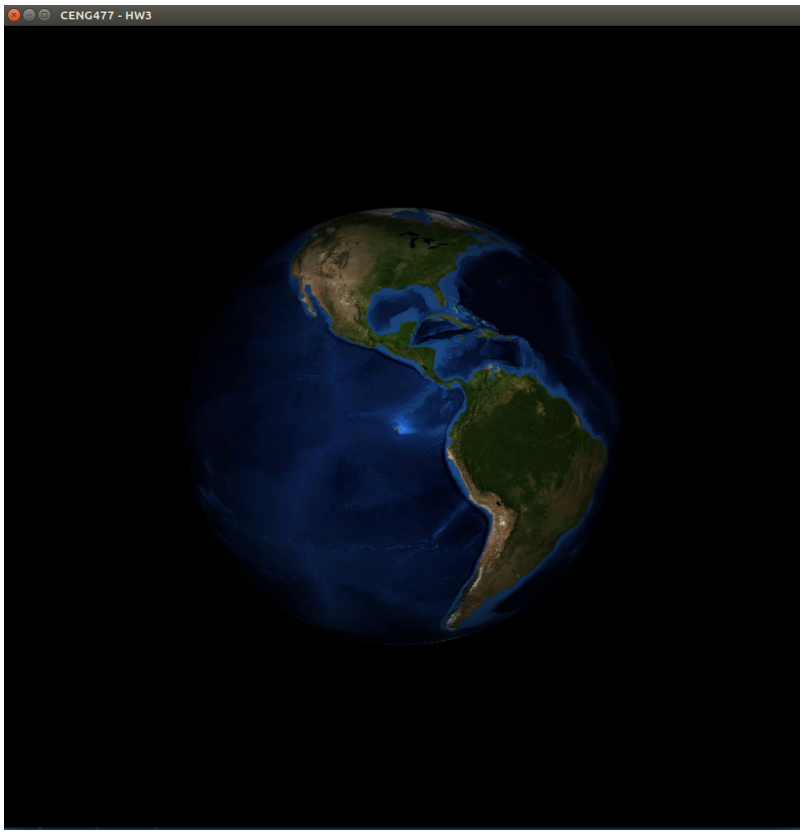
4. The keys used for moving or changing the position of elements inside the scene should be handled continuously. Namely, it should not be used with the action of any type; while pressing the key, it should execute the related commands repetitively.
5. The pixel value retrieved from texture map will have the coordinate of $(\frac{horizontal\_step}{horizontal\_split\_count}, \frac{vertical\_step}{vertical\_split\_count})$.
6. Heights of the vertices will be handled as described in 2.1.5. The only difference is that the direction of the height will be computed with vertex normal. Additionally, the initial height factor will be 0.

7. Q and E keys will move the texture and height map along the plane left and right respectively by subtracting and adding 1.
8. There will be one light source in the scene at the initial position of $(0, 1600, 0)$. The intensity of the light source is (1.0, 1.0, 1.0) and there will be no attenuation.
9. The light source can be moved using arrow keys. The height of the light is increased and decreased with T and G keys respectively. The position of the light source is changed by 5 for all directions.
10. Computing the surface color will be done as explained in 2.1.9, 2.1.10 and 2.1.11.
11. Camera will be implemented like described in 2.1.12. However, inital gaze vector will be $(0, -1, 0)$, initial position will be $(0, 600, 0)$ and up vector will be $(0, 0, 1)$.
12. When pressing the I key, the plane will be placed to the initial position with initial configurations of the camera and speed of 0.
13. Window and projection configurations will be the same as in 2.1.14 and 2.1.15.

3 Sample input/output

The sample input files are given at OdtuClass. The initial view of the OpenGL display window for the "height_gray_mini.jpg" and "normal_earth_mini.jpg" for flat and "height_gray_med.jpg" and "normal_earth_med.jpg" for sphere inputs are shown in the images below.





- Flat Earth video
- Spherical Earth video

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_flat
   >_ ./hw3_flat <height_map_file_name> <texture_map_file_name>
   >_ make hw3_sphere
   >_ ./hw3_sphere <height_map_file_name> <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.