Middle East Technical University | Department of Computer Engineering

CENG 477

Introduction to Computer Graphics

Fall '2016-2017
Assignment 4 - Rasterizing with OpenGL

Due date: December 25, 2016, Sunday, 23:59

1 Objectives

In this assignment, you are going to implement an OpenGL program to render a given scene and fly through the scene interactively by processing keyboard input. The input files for this assignment are very similar to ones in the first three assignments. A number of point light sources will be provided in the scene file. A number of material properties will be defined and assigned to triangular mesh models. The scene will be rendered using the built-in OpenGL illumination model and the smooth shading mode (see OpenGL related information at the end of the document). The parameters of a perspective view camera will be specified similar to the first three assignments. You will perform modeling transformations to triangular mesh models as in assignments #2 and #3. Since this will be your first interactive graphics experience, you will also implement a camera fly-through model by processing keyboard input.

Keywords: *forward rendering pipeline, OpenGL, modeling transformations, camera transformations, interactive fly-through*

2 Specifications

1. You should name your executable as "hw4".
2. Your executable will take two text files as argument: one that describes the scene and one that describes the cameras. The format of the camera file is very similar to the format of the camera file in the third assignments. The first difference is: there will be only one camera in the camera file. The second difference is: an output file name is no longer provided, since an interactive OpenGL display window will be used. The format of the scene file is described in the following section. You should be able to run your executable via the command "./hw4 sample_scene.txt sample_camera.txt".
3. The scene will only be composed of instances of triangles. A set of triangles will comprise a model, and as in the second and third assignments with a sequence of several transformations

(translation, rotation, scaling) you will be able to move, rotate, or resize a model (i.e., all of the triangles in the model). Transformations will be applied to the models in the order specified in the input file. You may use OpenGL functions to transform your models.

4. You will compute normal vectors of vertices as the average of the normal vectors of the triangles incident to this vertex.

5. You will use the Vertex Buffer Objects to send the position and normal data of the vertices to the GPU (using GL_ARRAY_BUFFER type buffer objects). Since, the data provided in the scene file is an indexed face format, you will also send triangles' vertex index data to the GPU (using GL_ELEMENT_ARRAY_BUFFER type buffer objects).

6. Models will be rendered in solid or wireframe modes as specified in the input file. Use the OpenGL function call `glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);` for the solid and the call `glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);` for the wireframe modes, respectively.

7. You will implement an interactive fly-through viewing functionality with the following specifications for each press of the following characters on the keyboard:

   w : Go forward in the current gaze direction 0.05 units. The gaze vector is unchanged.

   s : Go backward in the current gaze direction 0.05 units. The gaze vector is unchanged.

   a : Look Left. Rotate the gaze vector 0.5 degrees around the global y-axis, i.e., (0,1,0). The gaze, up, and u vectors of the camera should be updated after this rotation.

   d : Look Right. Rotate the gaze vector -0.5 degrees around the global y-axis, i.e., (0,1,0). The gaze, up, and u vectors of the camera should be updated after this rotation.

   u : Look Up. Rotate the gaze vector 0.5 degrees around the current u vector, (i.e., the x-axis) of the camera coordinate system. The gaze and up vectors of the camera should be updated after this rotation.

   j : Look Down. Rotate the gaze vector -0.5 degrees around the current u vector, (i.e., the x-axis) of the camera coordinate system. The gaze and up vectors of the camera should be updated after this rotation.

8. Lighting computations will be done by calling appropriate OpenGL functions (See Section 6 of this document).

9. You should enable the depth test to activate the depth buffer algorithm.

10. Use the smooth shading mode of OpenGL by calling the function `glShadeModel(GL_SMOOTH);`.

11. Unlike the first three assignments, the camera description file will contain only one camera and you may use glFrustum or gluPerspective functions to specify camera parameters in OpenGL.

3 Scene File

- **Background Color** R G B
- **Ambient Light** R G B
- **Number of point light sources** Integer
    - · **Position of light #i X Y Z**
    - · **Intensity if light #i R G B**
- **Number of materials** Integer
    - · **Ambient reflectance of material #i R G B**
    - · **Diffuse reflectance of material #i R G B**
    - · **Specular reflectance and exponent of material #i R G B SpecExp**
- **#Translations**
- **Translation Count** Integer
    - · t_x t_y t_z
- **#Scalings**
- **Scaling Count** Integer
    - · s_x s_y s_z
- **#Rotations**
- **Rotation Count** Integer
    - · α u_x u_y u_z
- **Number of vertices** Integer
- **#Vertices** · **Position of vertex #i X Y Z**
- **Number of meshes** Integer
    - · **Mesh Type** {1: solid, 2: wireframe}
    - · **Material ID** Integer
    - · **Number of transformations** Integer
        - · **Transformation**$_{type}$ **Transformation**$_{id}$
    - · **Number of triangles** Integer
        - · **Vertex index**$_1$ **Vertex index**$_2$ **Vertex index**$_3$

1. Background Color
Specifies the R, G, B values of the background as float values between 0 and 1.

2. Ambient Light
Specifies the R, G, B values of the ambient light intensity as float values between 0 and 1. Apply these values to the ambient component of all the light sources defined in OpenGL, e.g., GL_LIGHT0, GL_LIGHT1, etc.

3. Number of lights
Specifies how many point light sources are in the scene.

4. Light position
Specifies the the position of light source #i in the world coordinates.

5. Light intensity
Specifies the intensity of each light source as R G B values between 0 and 1. Apply these values to the diffuse and specular components of the light source.

6. Number of materials
Specifies how many materials will be defined.

7. Ambient
Specifies the ambient reflectance coefficients of material #i.

8. Diffuse
Specifies the diffuse reflectance coefficients of material #i.

9. Specular
Specifies the specular reflectance coefficients and the specular exponent of material #i.

10. Translation Count
Specifies how many translations are defined in the scene file.

11. Translation parameters
t_x, t_y, and t_z are the translation parameters, i.e., translation amounts along the major axes.

12. Scaling Count
Specifies how many scale transformations are defined in the scene file.

13. Scaling parameters
s_x, s_y, and s_z are the scaling parameters, i.e., scaling level in the corresponding coordinate axes.

14. Rotation Count
Specifies how many rotations are defined in the scene file.

15. Rotation parameters

α, u_x, u_y, and u_z are the rotation parameters, i.e., the object is rotated α degrees around the rotation axis which pass through points (u_x, u_y, u_z) and $(0, 0, 0)$. The positive angle of rotation is given as the counter-clockwise rotation along the direction (u_x, u_y, u_z).

16. Number of vertices
Specifies how many vertices are in the scene.

17. Positions
Specifies the position of each vertex in X Y Z starting with the vertex id 1.

18. Number of meshes
Specifies how many meshes (i.e., sets of vertices) are in the scene file.

19. Mesh Data

1. Model type of the mesh.1 for solid, 2 for wireframe.
2. Material id assigned to the mesh.
3. Number of transformations.
4. The 't' indicates a translation transformation, 's' indicates a scale transformation, and 'r' indicates a rotation transformation. The transformation id ranges from [1..Number of Translations] for type 't' transformations, and similarly for the other type of transformations.
5. Number of triangles.
6. Each triangle is given as a list of vertex ids in counter clockwise order when faced from the front side. Vertex ids start from 1.

4 Camera File

· **Position** X Y Z

· **Gaze** X Y Z

· **Up** X Y Z

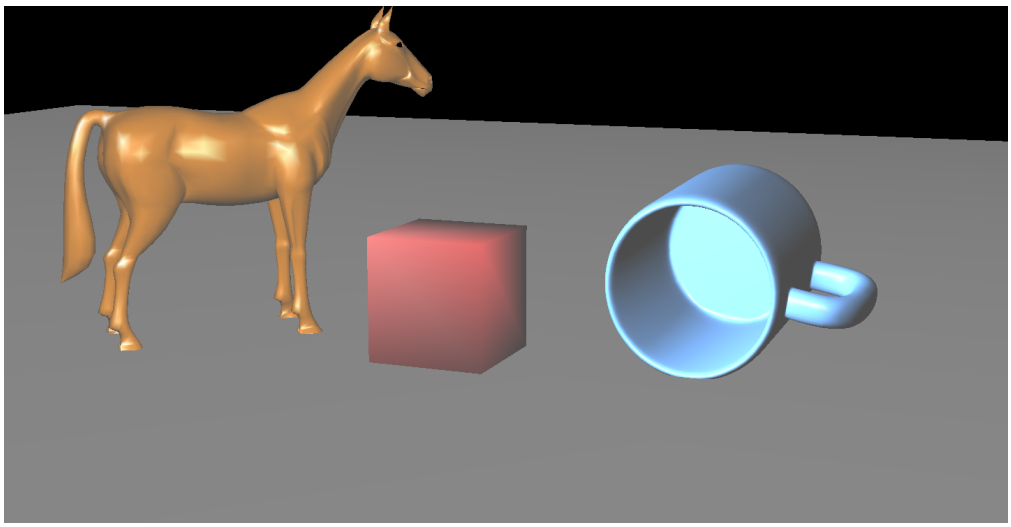
· **ImagePlane** Left Right Bottom Top Near Far HorRes VerRes

Position defines the X, Y, Z coordinates of the camera
Gaze defines the direction that the camera is looking at
Up defines the up vector of the camera. **The up vector is not necessarily given as orthogonal to the gaze vector. OpenGL's gluLookAt function automatically corrects the up vector.**
ImagePlane defines: the coordinates of the image plane in **Left**, **Right**, **Bottom**, **Top** parameters; distance of the image plane to the camera in **Near** and distance of the far plane to the camera

in **Far** parameters, and the size of the OpenGL window in **HorRes** and **VerRes** parameters. All values are floats except **HorRes** and **VerRes**, which are integers. You may assume that the **Gaze** vector of the camera is always perpendicular to the image plane.

5 Sample input/output

A sample camera and scene file is given at the following URL:
http://user.ceng.metu.edu.tr/ tcan/ceng477/20161/hw4/
The initial view of the OpenGL display window for these inputs is shown in the image below.



6 Additional OpenGL Related Information:

Lighting: To enable lighting computations, you should add the following code to your initialization function (i.e. it is sufficient to enable these only once).

```
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
glEnable(GL_LIGHT1);
.....
```

To specify the color and the position of each light source, add the following code to your initialization function:

```
GLfloat lightColor [] = {1.0, 1.0, 1.0, 1.0};
GLfloat ambientColor [] = {0.1, 0.1, 0.1, 1.0};
GLfloat lightPos []= {0.0, 20.0, 30.0, 1.0};
glLightfv(GL_LIGHT0,GL_POSITION,lightPos);
glLightfv (GL_LIGHT0,GL_AMBIENT, ambientColor);
glLightfv (GL_LIGHT0, GL_DIFFUSE, lightColor);
glLightfv (GL_LIGHT0, GL_SPECULAR, lightColor);
.....
```

The values above are just given as examples. For light position, light color (i.e., intensity), and the ambient light intensity, you will use the values provided in the input file. In addition, the above code only handles the first light source. You should write additional code for the other light sources (GL_LIGHT1, GL_LIGHT2, and GL_LIGHT3).

In OpenGL, light sources are not visible physically but they illuminate the surfaces of objects in the scene. Although not visible, a light source is transformed as other objects in your scene; so, you should make sure that its position is fixed by calling the `glLightfv(GL_LIGHT*,GL_POSITION,lightPos);` functions in your display function just after camera transformation. In other words, your display function should be structured as follows:

```
<clear the frame buffer>
<set viewport and projection parameters>
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
<camera transformations>
<set light position using the glLightfv function above>
<apply modeling transformations and draw models>
```

When lighting is enabled, OpenGL does not use glColor to color the surfaces by default. Instead, surface material properties are defined using the glMaterialfv function. To set the color of surface (or the entire model), use the following piece of code just before drawing the object.

```
GLfloat ambColor[4] = {<ar>, <ag>, <ab>, 1.0};
GLfloat diffColor[4] = {<dr>, <dg>, <db>, 1.0};
GLfloat specColor[4] = {<sr>, <sg>, <sb>, 1.0};
GLfloat specExp[1] = {<specular exponent>};
glMaterialfv(GL_FRONT,GL_AMBIENT,ambColor);
glMaterialfv(GL_FRONT,GL_DIFFUSE,diffColor);
glMaterialfv(GL_FRONT,GL_SPECULAR,specColor);
glMaterialfv(GL_FRONT,GL_SHININESS,specExp);
```

Finally, to determine the amount of light reflected from the surfaces correctly, you should specify the normal vectors of the vertices as Vertex Buffer Objects along with vertex coordinates.

7 Hints & Tips

1. Start early!
2. On Linux or on Windows, you need the definition of the extension library functions. They are provided in GLee (GLee.c and GLee.h) as additional files in COW (you may also use GLEW as an alternative).

   If you use GLee, in Linux you should compile and link the programs like the following:

   gcc -c -o GLee.o GLee.c gcc -g -o hw4 hw4.c GLee.o -lGL -lGLU -lglut

3. You must provide a makefile to compile your program. This file must use g++ or gcc as the compiler. You are free to choose your compile options. Failing to provide a working makefile will have a penalty of 5 points out of 100.

4. You may use -O2 option while compiling your code for optimization.

5. For debugging purposes, consider using simple scenes. Also it may be necessary to debug your code by tracing what happens for a single triangle (always simplify the problem when debugging).

8 Regulations

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

2. **Late Submission:** You can submit your codes up to 3 days late. Each late day will incur a penalty of 10 points. After 3 days, you will get 0.

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. 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 newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.

5. **Submission:**Submission will be done via COW. Create a tar.gz file named "hw4.tar.gz" that contains all your source code files and a makefile. The tar file should not include any subdirectories. The executable should be named "hw4" and should be able to be run using command "./hw4 scene_name.txt camera_name.txt". If your 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 5 points.

6. **Evaluation:** Your codes will be evaluated based on several input files including, but not limited to the test cases given to you as example.