Middle East Technical University | Department of Computer Engineering

CENG 477

Introduction to Computer Graphics

Fall 2014-2015

Programming Assignment 4 - Shadow Mapping in OpenGL Shading Language

Due date: 04 January 2015, Sunday, 23:55

1 Objectives

In this assignment, you are expected to implement shadows using OpenGL Shading Language. Scene will be constructed according to given files describing the scene properties. Details of files will be given in the specifications part. You are expected to correctly render the scene and shadows with the shadow mapping technique; you are also expected to implement keyboard interactions to change the position of the light, position of the the camera and rotation of the object. Up to 30% points you can gain extra points by implementing bonus parts.

Keywords: *GLSL, Shadow Mapping*

2 Specifications

Details of .obj files, input file, shadow mapping and keyboard interactions will be given in this section.

2.1 .obj File Format

OBJ file format is easy-to-use data format representing 3D models by their vertex positions, UV coordinates of each vertices, vertex normals and face informations. Because vertices are stored in a counter-clockwise, you don't need to declare face normals in .obj files. These normals can be calculated within the code.

There are four markers used in the .obj file to define attributes of vertices mentioned above :

- **v** x y z [w] : Vertices are specified with marker "**v**". That marker is followed by x y z [w] which are the x, y, z position of the vertex, respectively. w is optional and default value is 1.0.
- **vt** u v [w] : Texture coordinates of vertices are specified with marker "**vt**". These values vary between 0 and 1. w is optional and default value is 0.
- **vn** x y z : Normals of vertices are specified with marker "**vn**". Be careful, normal values might not be unit.

- Faces are defined using the indices of vertices, normals and textures. Marker of the face definition is "**f**". Faces can be specified by four different ways.

  f v1 v2 v3 : Indices of counterclock-wise vertices of faces are given by v1, v2, v3, respectively.

  f v1/t1 v2/t2 v3/t3 : Indices of counterclock-wise vertices and texture coordinates are given by v1/t1, v2/t2, v3/t3 pairs.

  f v1/t1/n1 v2/t2/n2 v3/t3/n3 : Indices of counterclock-wise vertices, texture coordinates and normals are given by v1/t1/n1, v2/t2/n2, v3/t3/n3 triples.

  f v1//n1 v2//n2 v3//n3 : Indices of counterclock-wise vertices and normals are given by v1//n1, v2//n2, v3//n3 pairs.

All the sample input files, except bonus scenes, consists of vertices and faces specified only with vertex indices. You can find the sample .obj file in the attachments of the homework. Simple .obj loader is also shared on the COW. If you want to implement **Texturing the Object** bonus. You should implement your own .obj loader functions.

2.2 Input File

In the scene you will render, there will be a box, stage model under the object model, object model. .obj files of the stage and object will be given as seperate files. Vertex positions of stage and object models are given in the objects space in .obj files. Lighting of the stage and object should be done according to material informations given in the input file. Phong shading must be implemented. Translation, rotation and scaling parameters of models should be implemented in the object space. Box coordinates will be given in the world space. Camera should always look to the (0,0,0) in the world space. The input file describing the scene will be in the following format.

#Light
f1 f2 f3 [Light position as 3 floats]
f1 f2 f3 [Ambient coefficient of Light as 3 floats]
f1 f2 f3 [Diffuse coefficient of Light as 3 floats]
f1 f2 f3 [Specular coefficient of Light as 3 floats]

#Camera
f1 f2 f3 [Camera position, camera must look at the (0,0,0) in the world space]
f1 f2 f3 [Camera up vector]

#MaterialStage [Material information of stage]
f1 f2 f3 f4 [Ambient coefficient of stage material as 4 floats]
f1 f2 f3 f4 [Diffuse coefficient or stage material as 4 floats]
f1 f2 f3 f4 [Specular coefficient of stage material as 4 floats]
f1 [Specular exponent of stage material]

#MaterialObject [Material information of object]
f1 f2 f3 f4 [Ambient coefficient of object material as 4 floats]
f1 f2 f3 f4 [Diffuse coefficient or object material as 4 floats]
f1 f2 f3 f4 [Specular coefficient of object material as 4 floats]
f1 [Specular exponent of object material]

#MaterialBox [Material information of box]
f1 f2 f3 f4 [Ambient coefficient of box material as 4 floats]
f1 f2 f3 f4 [Diffuse coefficient or box material as 4 floats]

f1 f2 f3 f4 [Specular coefficient of box material as 4 floats]
f1 [Specular exponent of box material]

#Box
x1 y1 z1 [Coordinates of the first vertex]
x2 y2 z2 [Coordinates of the second vertex]
. .
. .
x8 y8 z8 [Coordinates of the eighth vertex]
d1 [Number of triangles in the box]
a1 b1 c1 [Vertex indices for the first triangle as integer]
a2 b2 c2 [Vertex indices for the second triangle as integer]
. .
. .

#StageFile
nameOfStageFile [.obj file name of the stage]

#ObjectFile
nameOfObjectFile [.obj file name of the object]

#StageTransformation
x1 y1 z1 [Translation for the stage as float]
x1 y1 z1 [Rotation around x axis, around y axis, around z axis as float]
x1 y1 z1 [Scale coefficient for the stage as float]

#ObjectTransformation
x1 y1 z1 [Translation for the object as float]
x1 y1 z1 [Rotation around x axis, around y axis, around z axis as float]
x1 y1 z1 [Scale coefficient for the object as float]

2.3 Shadow Mapping

You are expected to implement shadows in this homework. You will render your scene and illuminate it correctly using GLSL. To make the scene more realistic, you should also implement shadows using shadow mapping technique with OpenGL programmable pipeline. Your implementation should create correct shadows regarding the position of the light source(s) and object. So, we would like you to correctly cast shadows of the loaded models onto the box and onto the stage. You can assume that light sources are directional, therefore rendering the shadow map should be done with an ortographic projection matrix.

2.4 Keyboard Interaction

The created scene will be interactive. Using keyboard input; position of the light source, position of the camera, and orientation of the model could be changed. Light position should be changed in the view space. Camera position should be changed in the model space. Object orientation should be changed in the object space. Predefined inputs are as follows:

- D or d : Move the light source in the +x direction.
- A or a : Move the light source in the -x direction.
- W or w : Move the light source in the +y direction.

- S or s : Move the light source in the -y direction.
- T or t : Move the light source in the +z direction.
- Y or y : Move the light source in the -z direction.
- O or o : Rotate the model and stage around the y axis in counterclockwise direction.
- P or p : Rotate the model and stage around the y axis in clockwise direction.
- Up arrow : Move the camera in the +y direction.
- Down arrow : Move the camera in the -y direction.
- Right arrow : Move the camera in the +x direction.
- Left arrow : Move the camera in the -x direction.
- N or n : Move the camera in the +z direction.
- M or m : Move the camera in the -z direction.

3 Bonuses

Extra efforts will be rewarded in this homework. You can earn up to 30% points.

- **Implementing faster rendering :** Faster implementations of the assignment will be rewarded by bonus points (max 10%). Bonus points will be calculated according to histogram of all implementations. FPS counter code will be shared on the COW.
- **Implementing two lights :** If you put an extra light on your screen you will also gain extra points (max 10%). That light can be stable inside the box, you should draw stable light with different color than movable light.
- **Texturing the object :** You will earn extra poins if you implement texture mapping on the object (max 10%). In order get texture information from the .obj file you should search for the details of the .obj format.

4 Notes

- If you have any error while compiling your code on your own machines, please use ineks.
- Shader loading and basic .obj file loading operations are implemented in the given template code.
- You are free to use any image loading library to load texture images.

5 Regulations

1. **Programming Language:** C/C++, GLSL
2. **Late Submission:** You can submit your codes up to 3 days late. Each 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.
4. **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.

6 Submission

Submission will be done via COW. Create a tar.gz file named `hw4.tar.gz` that contains all your C++ code, GLSL code and .obj files. Given library codes should also be included in your bundle. The following command sequence is expected to run your program on ineks:

```
$ tar -xf hw4.tar.gz
$ make
$ ./hw4 input.txt
```