Middle East Technical University Department of Computer Engineering

CENG 477

Introduction to Computer Graphics

Fall '2016-2017
Assignment 3 - A Software Rasterizer implementing the Forward Rendering Pipeline

Due date: December 14, 2016, Wednesday, 23:59

1 Objectives

In this assignment, you are going to implement Modeling Transformation, Viewing Transformation, and Rasterization stages of the Forward Rendering Pipeline. Specifically, given a set of triangles and the position and the color attributes of all of their vertices, your goal is to render the scene as a two-dimensional image. The parameters of a perspective view camera will be specified as in the first two assignments. You will transform all of the vertices to the viewport and then use line drawing and triangle rasterization algorithms to display the triangles in wireframe or solid modes. You will also implement the backface culling (for both modes) and the depth buffer algorithms for correct rendering of the scene. You must use C/C++ in your implementations.

Keywords: *forward rendering pipeline, modeling transformations, viewing transformations, line drawing, triangle rasterization, interpolation, depth buffer, backface culling*

2 Specifications

1. You should name your executable as "rasterizer".
2. Your executable will take two txt files as argument: one that describes the scene and one that describes the cameras (e.g. "sample_scene.txt sample_camera.txt"). The format of the camera file is very similar to the format of the camera files in the first two assignments. The only difference is the addition of a single scalar for indicating the distance of the far plane from the camera. The format of the scene file is described in the following section. You should be able to run your executable via the command "./rasterizer 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 assignment 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.

4. You will not implement any lighting computations in this assignment. The color of each vertex will be provided as input and your goal will be to interpolate color along the edges and the surfaces of the triangles in wireframe and solid modes, respectively.

5. Use the midpoint algorithm to draw triangle edges and use the barycentric coordinates based algorithm to rasterize triangle faces.

6. In both wireframe and solid modes, triangles whose backface is facing the viewer will be culled.

7. You will implement the depth buffer algorithm to correctly draw triangles that are closer to the viewer over the triangles that are farther.

8. You will NOT implement any clipping algorithm and you may assume that the camera is positioned so that the whole scene is within the viewing volume.

9. As in the first two assignments, the camera description file may contain multiple camera configurations.

3 Camera File

- **Camera Count** Integer
- **Camera "Cid"** id
  - · **Position** X Y Z
  - · **Gaze** X Y Z
  - · **Up** X Y Z
  - · **ImagePlane** Left Right Bottom Top Near Far HorRes VerRes
  - · **OutputName** imagename.ppm

1. Camera Count
Number of cameras that will be used for producing images of the scene.

2. Camera
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. Therefore you should find the camera's x-axis by a cross product of the gaze and up vectors, then correct the up vector by a cross product of gaze and x-axis vectors of the camera. 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 resolution of the final image 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. **OutputName** is a string which is the name of the output image.

4 Scene File

- **Background Color** R G B
- **#Vertices**
- **Number of vertices** Integer
- #Colors
    - **Color of vertex #i R G B**
- #Positions
    - **Position of vertex #i X Y Z**
- **#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
- **#Models**
- **Number of models** Integer
- **Model ID** Integer
    - **Model Type** {1: solid, 2: wireframe}
    - **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.

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

3. Colors
Specifies the color of each vertex in R G B starting with the vertex id 1.

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

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

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

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

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

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

10. 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).

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

12. Model ID

1. Model type of the model.1 for solid, 2 for wireframe.
2. Number of transformations.
3. 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.
4. Number of triangles.
5. 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.

5 Sample input/output

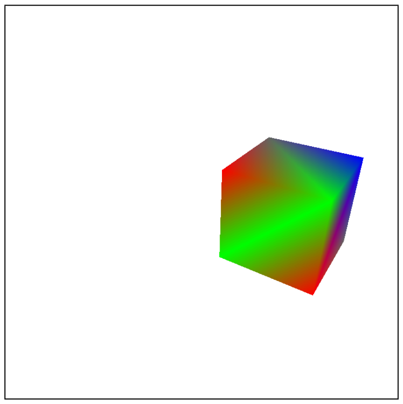
A sample camera and scene file and the corresponding output is provided below:
sample_camera.txt:

```
1
#Camera 1
0 5 0
0.1 -0.3 -0.5
0 1 0
-1 1 -1 1 2 1000 700 700
output.ppm
```

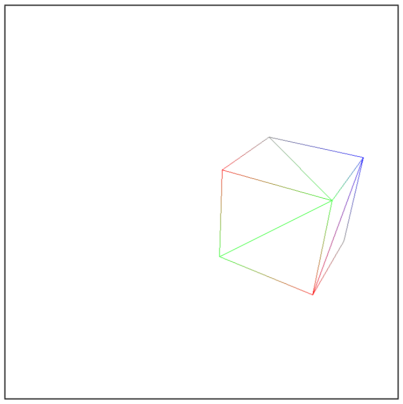
sample_scene.txt:

```
255 255 255
#Vertices
8
#Colors
100 100 100
255 0 0
0 255 0
0 0 255
0 0 255
0 255 0
255 0 0
100 100 100
#Positions
1.0 1.0 -1.0
-1.0 1.0 -1.0
-1.0 1.0 1.0
1.0 1.0 1.0
1.0 -1.0 -1.0
-1.0 -1.0 -1.0
-1.0 -1.0 1.0
1.0 -1.0 1.0
#Translations
2
0.0 10.0 0.0
3.0 -3.0 -6.0
#Scalings
1
5.2 5.2 5.2
#Rotations
3
45 0.0 1.0 0.0
60 0.8 0.6 0.0
20 1.0 0.0 0.0
```

```
#Models
1
1
1
3
r 1
t 2
s 1
12
7 8 4
7 4 3
8 5 1
8 1 4
6 3 2
6 7 3
3 4 1
3 1 2
6 2 5
2 1 5
5 8 6
7 6 8
```



The same input with the only change in the model type is:



6 Hints & Tips

1. Start early!
2. Do not forget to interpolate the depth (i.e, the z-coordinates) along the edges (in the wireframe mode) or faces (in the solid mode) of the triangles.
3. Note that in the wireframe mode, only the edges of the triangles will be drawn. For example, if there are two front facing triangles with the small triangle behind the large triangle, the small triangle will be visible in the wireframe mode, but will not be visible in the solid mode.
4. You must provide a makefile to compile your program. This file must use g++ 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.
5. You may use -O2 option while compiling your code for optimization.
6. 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).

7 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. To prevent cheating in this homework, we also compare your codes with online ray tracers 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 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 "rasterizer.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 "rasterizer" and should be able to be run using command "./rasterizer 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. Rendering all scenes correctly will get you 100 points.