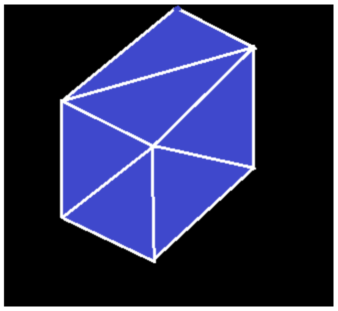
Homework #4: Shaded Rendering I

Due Date: 11:59pm Monday 28 October 2013



In this homework you will convert your wireframe renderer into a shaded renderer, where the triangles will be filled with solid color and outlined with another color.

Scene Description

The input language used is the same OpenInventor language from Homework #3.

Rendering

For this homework you will need to rasterize triangles using the method described in the lectures (using Barycentric coordinates). You will need to implement Barycentric interpolation to find out if a pixel is inside a triangle or not, and also to find its depth z-value given the z-values of the vertices.

Hidden Surface Removal

You will implement a simple Z-buffer for your renderer. Whenever you draw a pixel (given its x and y position plus its depth), you have to make sure that its depth is smaller than the current value in the Z-buffer, otherwise this pixel is discarded.

Input and Output

Your program should be called "*shaded*" and should take as input the resolution in pixels (xRes and yRes)

The scene description is given in the format described above on stdin and your program should produce a PPM image file on stdout.

```
shaded xRes yRes < input.iv | display -
```

The parser is given in the directory "*shaded*", and it has a sample program called "`shaded.cc`" which can read input from stdin and parse it. Make sure you can compile and run it using the supplied Makefile:

```
make all
./shaded < ../data/cube2.iv
```

Program Flow

The flow of the program should be as follows:

- Read the input describing the object points, faces, and transformations (using the provided parser).
- Loop on the faces of the object, and for each face:
    - Convert the vertices to pixel coordinates
    - Draw lines through the vertices using the Midpoint algorithm making sure to **connect** the **first** and **last** points. Use **white** color for the edges.
    - Rasterize the triangle by coloring all pixels belonging to the triangle with **blue**. Make sure to interpolate the z-value of the pixels to implement the Z-buffer correctly.
- Output the drawn lines in PPM format as in Homework #1.

The transformation to the points to convert them from the object space to the pixel space is the same as Homework #3.

Instructions

- All code should be implemented in C++ under Linux.
- Please submit your homework in one zip file named as follows: *HW##.FirstName.LastName.zip*, so for example if your name is Mohamed Aly and this is homework #1, then the file name should be *HW01.Mohamed.Aly.zip*.
- Please include all your code and sample output in the zip file, with a README file to explain what you did. Failure to follow these instructions will cause deductions from your grade.
- You are allowed to discuss the problems among yourselves. However, **copying** any part of the code will result a grade of **ZERO**. No exceptions.

Acknowledgment

This homework is adapted from CS 171 at Caltech.