

CS535 Fall 2004 Programming Assignment 2

Z-Buffering and Shading

The goal of this assignment is to build upon what you did in assignment 1, filling in your triangles while correctly handling hidden surface removal (visible surface determination). Again, OpenGL should only be used to draw pixel by pixel. In the new version of the program, one of the floating cubes should be changed to a sphere (a mesh approximating a sphere). You should add colors to your data structures.

As an option, you will also add shading to your triangles. You then need normals and a light source in your data structures. The scene should then be rendered using Gouraud shading.

Suggested steps, baseline:

1. Change one of the floating cubes into a sphere. This is most easily done using the same parameterization of the sphere that you already used for the center of projection of the camera given r , α and β . Set $r=1$. Then use the parameterization over the intervals $0 \leq \alpha < 2\pi$ and $-\pi/2 \leq \beta < \pi/2$. Add the resulting x, y, z values to the center of the sphere to get a vertex location in the world. Use a mesh like the one on the ground plane to tessellate the (α, β) -space.
2. Insert code that allocates (and deallocates) a z-buffer containing floating point values. The buffer should store the $1/z$ values for each pixel in the drawing window. Note that this means that re-allocation needs to occur when the window is resized.
3. In the beginning of the drawing function, insert a loop that clears the z buffer (sets $1/z=0$).
4. Insert code that scan-converts a triangle. (see Appendix).
5. In the scan-conversion function, use the $1/z$ values at the vertices and interpolate linearly over the triangular patch. Check against the previous value stored in the depth buffer. Only draw when $1/z$ is larger than the previous value. When it is, update the $1/z$ value and the color.
6. Add color to your data-structures. Do this such that the sides of the cube and the squares on the ground-plane have uniform colors but distinct from their neighbors.

Optional:

7. Add linear interpolation of the color values from the vertices to the scan conversion function.
8. Compute normals for the triangle vertices in the mesh. Apart from the sphere, use a vector product and then normalize. Apart from the sphere you should use the same normals for all three vertices of a triangle. For the sphere, the x, y, z values that are used to create the vertex in a vector and normalize it to unit magnitude to obtain the surface normal at this vertex.
9. Insert a point light source at some world point, $(X, Y, Z) = (10, 10, 10)$, say.
10. Transform the point light source and the normals by the world to view transformation.
11. Compute the shading of each vertex using at least an ambient and a diffuse component (optionally also Phong) in the camera coordinate system and use it instead of the material color when passing the vertex shadings to the scan-conversion function.
12. Step the light source around on a circular trajectory when the user presses a key.

Appendix: Hints for Scan-converting a Triangle

To scan convert a triangle, first write a function that scan-converts the quadrangle area with the four corners (x_1, y_1) , (x_2, y_1) , (x_3, y_2) , (x_4, y_2) (see Figure 1). Write a loop that for each scan line linearly interpolates to obtain the start and end-points $(x, y, 1/z)$ on each scan-line. Linearly interpolate within the scan-line and draw using the $1/z$ buffer.

Then write the triangle scan-conversion function that takes the three image points (x_1, y_1) (x_2, y_2) (x_3, y_3) that are the projected vertices of the triangle. Sort the coordinates (and associated properties such as color and $1/z$) based on increasing y -coordinates. Split the triangle into two areas of the above type (you have to interpolate to get the coordinates of an additional point, see Figure 2) and scan convert both areas using the above function.

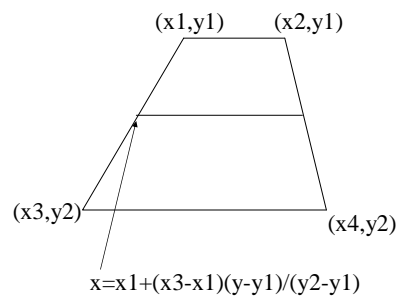


Figure 1. Scan-conversion of quadrangle.

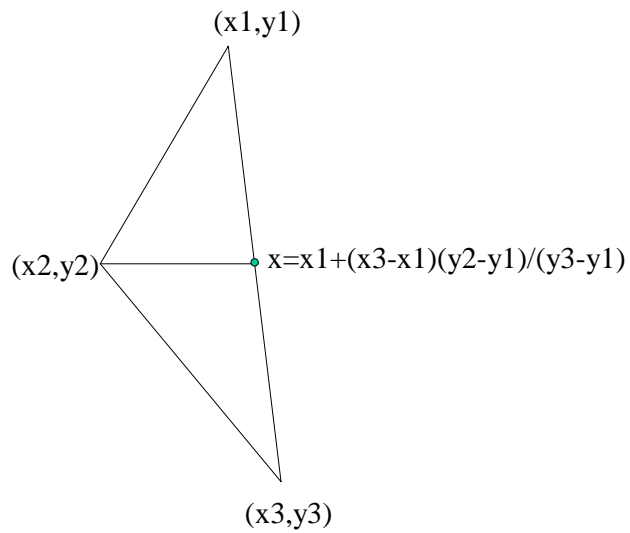


Figure 2. Splitting a triangle into two.