

CS535: Assignment #2 – Shaded Renderer

Out: September 13, 2005

Back/Due: September 29, 2005

Objective:

This focus of this assignment is to obtain a good understanding of transformation, rasterization, and shading. You have almost 2.5 weeks but it includes all your previous assignment efforts. In particular, this assignment builds upon the transformation library/object-classes you implemented for the previous assignment. You will write a program that rasterizes 3D objects to the screen, colors the pixels using shading calculations, and animates the object as in Assignment #0.

Summary:

The assignment is to implement a program which renders a solid, shaded 3D object bouncing within a box-shaped area. Using a GLUT-based user interface, allow the user to choose the size of the box (x-, y-, and z-dimensions), the object filename, the scale of the object (s), the position of the middle of the object (o_x, o_y, o_z), the rotation of the object about its mid-point (r_x, r_y, r_z), the object color (o_r, o_g, o_b), the object ambient reflection coefficient (a_o), the object diffuse reflection coefficient (d_o), the field of view in degrees (fov), a near distance (n), a far distance (f), the viewing position (v_x, v_y, v_z), the viewing direction (d_x, d_y, d_z), the position of the light source (l_x, l_y, l_z), the light intensity (l_i), the ambient light intensity (a_l) (you may assume it is white light), and diffuse light intensity (d_l) (also, you may assume it is white light). The scene is being constantly redrawn as the object bounces within the world box. Changes to any of the above parameters should be effective in the next drawn frame.

Specifics:

(0) Object Representation: each object should consist of a list of vertices and a list of triangles; thus any 3D object approximated by triangles can be used. You must provide files for at least a cube, pyramid, and an approximation of a cylinder. The file format to use for *all* objects is:

- a. n
- b. $v_{x0} v_{y0} v_{z0}$
- c. $v_{x1} v_{y1} v_{z1}$
- d. ...
- e. $v_{x(n-1)} v_{y(n-1)} v_{z(n-1)}$
- f. t
- g. $v_{a0} v_{b0} v_{c0}$
- h. $v_{a1} v_{b1} v_{c1}$
- i. ...
- j. $v_{a(e-1)} v_{b(e-1)} v_{c(e-1)}$

where n = number of vertices, t = number of triangles, v_{x0} is the x-coordinate of the first vertex, v_{a0} , v_{b0} and v_{c0} are the indices of the vertices of the first triangle, etc.; all indices start at 0.

(comment lines will be preceded by a '#' character)

- (1) User Interface: a GLUT-based user interface must be used to allow the user to specify all the parameters listed above. Reasonable initial values should be given to all parameters at program initialization. The default viewing position should be (0,0,0) and default viewing direction should be (0,0,-1). A reasonable field of view is 60 degrees. A good near and far distance is 0.01 and 10, respectively. A good box size is -1 to +1 in x, y, and z. Objects should fit within that box as well. A typical object size is a diameter of 0.1. By default, the object should be placed in front of the camera (e.g, down the $-z$ axis) and the scale factor should be $s=1$.
- (2) Transformations: you will have to use the library and/or object-classes from the previous assignment.
- (3) Object, camera, and perspective transformations: the same as with the previous assignment.
- (4) Clipping: for this assignment you will have to do basic clipping of objects. This means that points (pixels) on the object surface outside the field of view, closer than the near distance, or farther than the far plane should not be rendered.
- (5) Rasterization: you will have to implement triangle rasterization. This means you will have to transform the three coordinates of each triangle and then “fill-in” the interior pixels using a triangle rasterization algorithm (such as the barycentric-coordinate based method we explained in class).
- (6) Shading: you will have to compute the ambient and diffuse lighting equation for Lambertian (diffuse) surfaces at each vertex using the object and light properties ($I_r = I_i * a_i * (a_o * o_r) + I_i * d_i * (d_o * o_r) * (N \cdot L)$, $I_g = \dots$, $I_b = \dots$). This will produce three RGB colors for the triangle which you will have to interpolate across the surface of the triangle during rasterization (thus you will be doing Gouraud shading). Assume the triangle vertices are given in counter-clockwise order and that the cross product of triangle edges defines the triangle surface normal; e.g., if triangle has vertices (a,b,c), then the normal equals (b-a) x (c-a).
- (7) Object ordering: you do not want “farther away” triangles to overwrite closer-to-the-viewpoint triangles. Z-buffering is one common method. For this assignment, you will use a simpler method: painter’s algorithm. Simply sort the triangles from back to front for each frame and rasterize from the farthest triangle forward. Objects contain only triangles and are nicely connected so that sorting is simple.
- (8) Collisions and Bouncing: similar to Assignment #0, the object must bounce off the inside walls of the world environment but this time it is a 3D box and a 3D object. You must use a random (but reasonable) initial translational velocity vector and speed (such that each time you start the program, bouncing starts differently). Objects do not need to have an angular velocity. Collision response is thus simply a reflection of the translational velocity vector. You will have to compute penetration/collision of the collection of triangles with the wall. You may assume the object is convex, thus checking for vertex penetration into the wall is sufficient (note: be careful about collisions near the corner of the world box). As with Assignment #0, the object must bounce and reflect off the inside walls of the world box and must “touch” the wall before bouncing along the reflected direction: remember incident angle equals reflected angle.

Extra Credit:

- (1) Phong Shading

- (2) Implement additional lighting effects: attenuation by distance, atmospheric effects (e.g., fog), multiple lights, metal vs. plastic materials, etc.
- (3) Implement angular velocity and the appropriate collision response.

GLUI: (mandatory in this assignment)

GLUI is a platform independent user-interface builder. There are other similar libraries out there but I recommend this one for its simplicity yet completeness. You must install this library. More information about GLUI can be found at <http://www.cs.unc.edu/~rademach/glui>.

Grading:

Your program will be tested against the three object files you provide and additional object files we provide. We will use the interface to alter the aforementioned parameters and expect to see the correct behavior. If the interface does not allow a certain parameter to be changed, it will be considered not implemented at all. If the program does not compile, zero points will be given.

To give in the assignment, email the TA, Denny Wong (wang124@purdue.edu), your complete project (project files, source code and precompiled executable) by 4:30pm on the due date. It is your responsibility to make sure the email is delivered/dated before it is due. If you wish to receive confirmation of receipt, please ask the TA by email. If you email at 4:29:59pm on the due date and ask for confirmation and the confirmation is negative -- it will be considered late. **Hint: don't wait until the last moment to hand in the assignment.**

IMPORTANT: in your email message, please include a short message saying "This is Assignment #X" and attach your assignment inside a .ZIP file -- do not email a .EXE file as spam filters will probably filter-out the attachment.

If you have more questions, please see myself or the TA.