CS-411 – Assignment 4 (5%)

Surface Rendering

Due by: November 2, 2017

In this assignment you need to write a simple surface viewer. The object to be rendered is read from an input file containing vertex and face information. After rendering the object keyboard keys may be used to move the camera around it. Answer the following questions and implement the necessary code. Submit the assignment using the submission instructions of the first assignment. A skeleton program and sample object models will be provided. You are not required to use the skeleton program and may modify it as needed.

1. General questions (use JavaScript and/or manual computations):

   1.1 Write the transformation matrix for rotating a 3D point in homogeneous coordinates by 45^o about a rotation axis given by $(0, 1, 0)$.

   1.2 Write the transformation matrix for rotating the vector $(1, 1, 0)$ to become aligned with the y-axis $(0, 1, 0)$ in homogeneous coordinates.

   1.3 Write the transformation matrix for rotating a 3D point in homogeneous coordinates by 45^o about a rotation axis given by $(1, 1, 0)$. Express the transformation matrix you write in terms of the transformation matrices in the previous two questions.

   1.4 let $p = (1, 2, 3)$ be a desired location of an airplane. Let $v = (-1, -2, -3)$ be the desired direction of flight. Let $u = (1, 0, 1)$ be an up vector. Compute the transformation matrix that will position and orient an airplane according to the preceding specifications. Assume that initially the airplane is located at the origin and aligned with the x-axis.

   1.5 Let $r1 = [1, 2, 3, 3]$, $r2 = [2, 3, 4, 4]$, $r3 = [3, 4, 4, 5]$, $r4 = [0, 0, 0, 1]$ be the 4 rows of the transformation matrix M that is applied to the vertices of a 3D model. Let $v = (0, 0, 1)$ be a vector that is normal to one of the surfaces in the model. Compute the coordinates of the normal vector after the object is transformed by M. Recall that the normal is transformed differently from vertices and that the homogeneous coordinate should be ignored.

   1.6 Let $p = (1, 2, 3)$ be a 3D point in camera coordinates. Let $M_{\text{orthographic}}$ be an orthographic projection matrix that preserves the z-coordinate. Compute the coordinates of the point p after projection.

   1.7 Let $p = (1, 2, 3)$ be a 3D point in camera coordinates. Let M_{parallel} be an oblique parallel projection matrix that preserves the z-coordinate, whereas it is assumed that the direction of projection is given by $v = (0, -1, 1)$ and the the image plane is positioned at the origin. Compute the coordinates of the point p after projection.

   1.8 Let $p = (1, 2, 3)$ be a 3D point in camera coordinates. Let $M_{\text{perspective}}$ be a perspective projection matrix, whereas it is assumed that the focal length is given by $f = 10$, and that the image plane is perpendicular to the z-axis. Compute the coordinates of the point p after projection.

2. Program implementation (use JavaScript and WebGL):

   2.1 *Input file format:* The input text file describes one 3D object. Each line in that file describes either a vertex or a face. Empty lines, and lines beginning with a # should be ignored. The following are possible lines in the input file:

   ```
   # Specify a VERTEX:
   v <float-coord-x> <float-coord-y> <float-coord-z>

   # Specify a FACE (triangle):
   f <vertex-index-1> <vertex-index-2> <vertex-index-3>
   ```

   2.2 *Object loading:* Read vertex lines should be stored in a vertex array based on their order in the input file. Read face lines should be stored in a face array. Assume that all the faces are triangles. A face is specified by the indexes of three vertices in the vertex array.

   2.3 *Object normalization:* The coordinates of the loaded object should be scaled so that the larger side of its bounding box will have a length of one. The coordinates should be normalized so that the center of mass of the object will be placed at the origin. This may be achieved using a translation and scale transformations.

   2.4 *Vertex normals:* In order to obtain correct lighting of the object, for each drawn face (polygon) a normal vector should be specified. The normal vector is not provided and should be computed. First a normal vector should be computed for each face. Then a unique normal vector should be computed at each vertex by averaging the normals of the faces that intersect it: (i) Compute a normal for each face and add it the existing normal of each of its vertices. (ii) Once the complete list of faces is traversed, normalize all the vertex normals.

   2.5 *Moving the camera:* after rendering the object, the user should be able to rotate the camera about the object (left/right/up/down) and zoom in/out by using the keyboard or buttons. Instead of rotating the camera it is simpler to rotate the object. The way to handle rotation is to store the current rotation matrix and add to it by left multiplying it by a new desired rotation corresponding to a user command (i.e. left, right, up, down rotation).

   2.6 *Normal direction:* The vertex winding order in the provided objects is guarantied to be consistent within objects. It is not guaranteed to be consistent between objects. To identify whether the normals of an object need to be inverted due to a different winding order do the following: (i) Given a point X on the surface with normal N, and given the center of mass of the object C, verify that $N \cdot (X - C) > 0$. (ii) If this product is smaller than zero invert the normals. In addition add a keybord key or button to invert the current normals.

   2.7 *Computing vertex normals:* When the object file is read, each vertex is assigned the face normal. You need to replace these normals with average vertex normals. Here is how to do this:

   - Set all the normals in *model.arrays.normals* to 0.
   - Scan the face table, and for each face do:
     - retrieve the index of the 3 vertices that define the face (v0,v1,v2).
     - compute a face normal using cross product (e.g. N=(v2-v0)x(v1-v0))
     - Add N to entry v0 of *model.arrays.normals*
     - Add N to entry v1 of *model.arrays.normals*
     - Add N to entry v2 of *model.arrays.normals*
   - Scan the array *model.arrays.normals* and normalize each normal

3. Submit via bitbucket using the submission procedure of assignment 1.