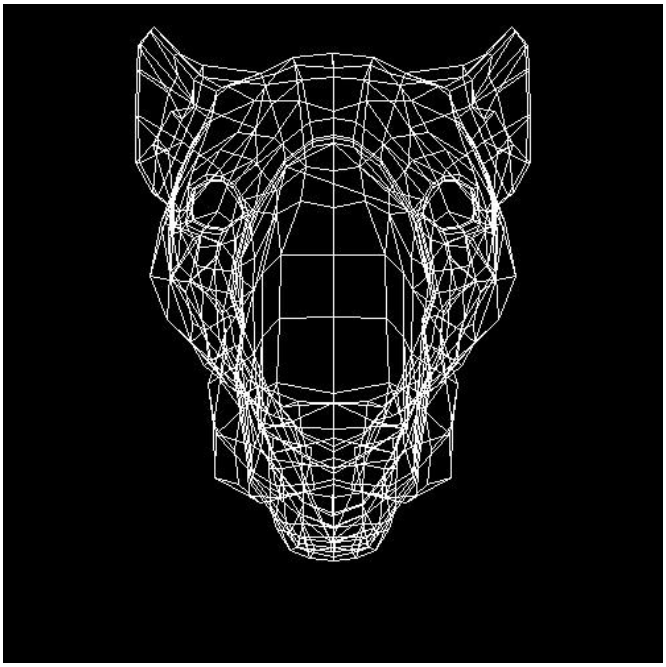
Homework #3

Wireframe Renderer



Write a C++ program to implement a wireframe renderer. It should read its input from stdin and put its output on stdout. The input will describe an object, defined by its faces, and the output should be the wireframe of the object i.e. drawing the faces (polygons) of the object by connecting lines through its vertices.

Your program will implement the viewing transformations pipeline i.e. transform from the object space to world space, then to the camera space, then apply the perspective projection onto the canonical view volume and finally to pixel coordinates, where lines are drawn using your implementation of the Midpoint algorithm from Homework #1.

Scene Description

The input language used is the OpenInventor language, for which you will be given a parser to help parse in the input for you.

```
PerspectiveCamera {
   position x y z
   orientation x y z theta
```

```
   nearDistance n
   farDistance f
   left l
   right r
   top t
   bottom b
}

# Zero or more lights
PointLight {
   location x y z
   color    r g b
}

# One or more Separator blocks
Separator {
   # One or more Transform blocks per separator
   Transform {
      translation tx ty tz
      rotation x y z theta
      scaleFactor sx sy sz
   }

   # One block per Separator
   Coordinate3 {
      point [
          x0 y0 z0,
          x1 y1 z1,
          ...
         xN yN zN]
   }

   # One block per Separator
   IndexedFaceSet {
     # Indices of vertices of the faces
     coordIndex [
       face0p0, face0p1, face0p2, face0pn, -1,
       face1p0, face1p1, … face1pn, -1,
       ...
       faceNp0, faceNp1, …. -1]
   }
}
```

The block "*PerspectiveCamera*" defines the perspective camera viewing the scene. It defines the c amera position in space and its orientation (rotation axis and angle). From *position* and *orientation* we can define the world to camera space transformation (M_{cam} from the inverse translation followed by the inverse rotation i.e. $M_{cam} = R^{-1}\ T^{1}$). Next the perspective view frustum is defined by its *nearDistance,*

farDistance, left, right, top, and *bottom* planes. From these values we can define the perspective projection matrix *P*.

The block "*Separator*" defines a logical placement for objects that have the same transformations and materials. It includes one or more "*Transform*" blocks, one "*Coordinate3*" block, and one "*IndexedFaceSet*" block.

The block "*Transform*" defines the transformation applied to the points to convert them from the object to the world space M_o. It consists of only one rotation, one translation, and one scaling in any order. If one of them is missing then it is assumed to be identity. The transformations have to be applied in the order: scaling then rotation then translation i.e. $M_o=TRS$.

Note: There can be more than one "*Transform*" block, in which case the top one is applied last and the bottom one is applied first to the object points. For example, if we have three Transform blocks T1 then T2 then T3, they are equivalent to one transformation $T = T1\ T2\ T3$ where T3 is applied first to the objects.

The block "*Coordinate3*" defines a number of 3D points which will make up the vertices of the faces of the object. The points are numbered starting at *zero*.

The block "*Normal*" defines a number of 3D normal vectors that will make up the normal vectors at the vertices, and numbering starts at *zero*.

The block "*IndexedFaceSet*" defines the faces of the object by defining the vertices. Indexing is done into the "*Coordinate3*" block. It has one sub-block, "*coorIndex*", that contains the indices of the vertices of each face, where faces are separated by a value of "**-1**". For example, 0 1 2 3 corresponds to a face whose vertices are the first four points and 1 3 5 6 corresponds to a face consisting of the points 1, 3, 5, and 6.

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
- 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 as follows:

- Apply the object transformation to convert into world space
- Apply the camera transformation to convert into camera space (using inverse translation

followed by inverse rotation of the camera).

- Apply the perspective transformation to convert into the canonical view volume
- Apply the viewport transformation to convert into pixel coordinates

Input and Output

The program should take the *xRes* and *yRes* on the command line arguments and read the scene description from stdin and outputs the PPM file with the rendered wireframe on stdout. For example:

```
wireframe 256 256 < scene.txt | display -
```

should output a 256x256 PPM file describing the scene in `scene.txt`.

Acknowledgment

This homework is adapted from CS 171 at Caltech.