B481/B581 — Computer Graphics (A. Hanson)

Interacting with 3D Objects.

- To rotate a point by an angle θ about the axis $\hat{n}$, where $\hat{n} \cdot \hat{n} = 1$, using the following matrix:

$$R_{ij} = \delta_{ij} \cos\theta + n_i n_j (1 - \cos\theta) - \epsilon_{ijk} n_k \sin\theta,$$

  where ϵ_{ijk} is the totally antisymmetric symbol in the indices.
  The explicit form of the matrix is

$$\left| \begin{array}{ccc} \cos\theta + (n_x)^2(1-\cos\theta) & n_x n_y(1-\cos\theta) - n_z \sin\theta & n_x n_z(1-\cos\theta) + n_y \sin\theta \\ n_y n_x(1-\cos\theta) + n_z \sin\theta & \cos\theta + (n_y)^2(1-\cos\theta) & n_y n_z(1-\cos\theta) - n_x \sin\theta \\ n_z n_x(1-\cos\theta) - n_y \sin\theta & n_z n_y(1-\cos\theta) + n_x \sin\theta & \cos\theta + (n_z)^2(1-\cos\theta) \end{array} \right| .$$

Object structures. Polyhedra can be read in using the following functions in `readoff.c`:

```
void ReadOFF(char *filename, Object3D *obj);
void InitObj(Object3D *obj);
void PrintObj(Object3D *obj);
void CopyObj(Object3D *src, Object3D *dst);
void FreeObj(Object3D *obj);
```

and are defined using this structure defined in `readoff.h` and allocated by `ReadOFF`:

```
typedef struct {
  Point3 center; /* world center
  int Nvertices;                /* number of total vertices */
  int Nfaces;                   /* number of total faces */
  HPoint3 * vertices;           /* coordinates of each vertex */
  int *nv_face;                 /* number of vertices for each face */
  int **faces;                  /* faces[i][j] is the index of face[i]'s
                                   j-th vertex */
} Object3D;
```

Example OFF file. This is an example of an OFF file readable by `readoff.c`:

```
# a simple OFF file for a tetrahedron
OFF                # header keyword
4 4 6               # NVertices Nfaces (ignored: Nedges)
# vertices: x y z
1.0 0.0 0.0
0.0 1.0 0.0
0.0 0.0 1.0
0.0 0.0 0.0
# faces: nface_verts vert_0 vert_1 ... vert_(nface_verts-1)
3  0 1 2
3  0 3 1
3  0 2 3
3  1 3 2
```

Perspective Assume the Z axis points at the camera, and and that the (u, v) film plane of the camera lies on the Z axis a distance $D - f$ from the world origin, with the focal center a distance f behind that, so the focal center is a distance D from the origin. Then the film plane coordinates of a point (x, y, z) on a polyhedron are

$$
\begin{aligned}
X &= \frac{fx}{(D-z)} \\
Y &= \frac{fy}{(D-z)} \,.
\end{aligned}
$$

Manual transforms can be done by performing a final transformation from this *perspective projected* (X, Y) space to a reasonable (u, v) screen space using a `WorldToDevice` function.

Eliminating Hidden Faces First determine the normals of each face, then take the dot product of the normal with the vector from any vertex $\vec{V}$ on the face to the camera focal center $\vec{C}$, that is, compute $\hat{n} \cdot (\vec{C} - \vec{V})$. If this is positive, draw the face, if the dot product is negative, do not draw the face.

The Rolling Ball For the Rolling Ball transformation, recall that we simply use the following notation:

$$
\begin{aligned}
n_x &= \frac{-dy}{dr} \\
n_y &= \frac{+dx}{dr} \\
n_z &= 0,
\end{aligned} \tag{1}
$$

where we define the input-device displacement $dr = (dx^2 + dy^2)^{1/2}$.

The single free parameter of the algorithm is the effective rolling ball radius R, which determines the sensitivity of the rotation angle to the displacement dr. (Try between 100 and 200 pixel units.) We choose the rotation angle θ to be

$$
\theta = \arctan \frac{dr}{R} \tag{2}
$$

so that

$$
\begin{aligned}
\cos\theta &= \frac{R}{(R^2 + dr^2)^{1/2}} \\
\sin\theta &= \frac{dr}{(R^2 + dr^2)^{1/2}}.
\end{aligned} \tag{3}
$$

Clearly, for small angles, we can also use $\theta \approx dr/R$.