Today’s class

- Orthogonal matrices
- Quaternions
- Shears
- Synthetic camera
- Viewing coordinates
- Projections
Arbitrary rotation review

- Recall the steps necessary to do a rotation about an arbitrary axis
  - Rotate $-\phi$ around $z$-axis to bring $L$ to $x$-$z$ plane
  - Rotate $-\nu$ around $y$-axis to bring $L$ to $z$-axis
  - Rotate $\theta$ around $z$-axis
  - Rotate $\nu$ around $y$-axis
  - Rotate $\phi$ around $z$-axis

- How easy is it to undo this transformation?
Inverse of a rotation

- Recall the rotation matrix in 2-D
  \[
  \begin{bmatrix}
  \cos \theta & -\sin \theta \\
  \sin \theta & \cos \theta
  \end{bmatrix}
  \]

- The inverse of a rotation rotates by the negative of the angle

  \[
  \begin{bmatrix}
  \cos(-\theta) & -\sin(-\theta) \\
  \sin(-\theta) & \cos(-\theta)
  \end{bmatrix}
  \begin{bmatrix}
  \cos \theta & \sin \theta \\
  -\sin \theta & \cos \theta
  \end{bmatrix}
  \]

- The product of these two matrices is the identity matrix
Orthogonal matrices

- An **orthogonal matrix** is a matrix where the rows and columns are mutually orthogonal unit-length vectors.
- Each of the simple axis rotation matrices is an example of an orthogonal matrix.
Properties of orthogonal matrices

- The inverse of an orthogonal matrix is equal to the transpose of the matrix.
- The product of two orthogonal matrices is an orthogonal matrix.
Quaternions

- Extensions of complex numbers
- An alternative approach to describing and manipulating rotations
- Provide advantages for animation and hardware implementation of rotation
Representation of a quaternion

- Recall the OpenGL command for rotation:
  \[ \text{glRotate} \{d,f\} \ (\text{angle, } x, y, z) \]
- The four parameters define a \textit{quaternion}, where the first item is a scalar and the last three form a vector
- \[ a = (q_0, q_1, q_2, q_3) = (q_0, \mathbf{q}) \]
Rotations via quaternions

- Let the quaternion $p = (0, p)$ represent a point in space.
- Define the quaternion $r = (\cos \frac{1}{2} \theta, \sin \frac{1}{2} \theta \mathbf{v})$, where $\mathbf{v}$ has unit length.
- $|r| = \cos^2 \frac{1}{2} \theta + \sin^2 \frac{1}{2} \theta \mathbf{v} \cdot \mathbf{v} = 1$
- $r^{-1} = (\cos \frac{1}{2} \theta, -\sin \frac{1}{2} \theta \mathbf{v})$
- Let $p' = rpr^{-1}$.
- This quaternion can be shown to have the form $(0, p')$, and thus represents a point.
- It is the result of rotating $p$ by $\theta$ about the vector $\mathbf{v}$. 

"Let the quaternion $p = (0, p)$ represent a point in space. Define the quaternion $r = (\cos \frac{1}{2} \theta, \sin \frac{1}{2} \theta \mathbf{v})$, where $\mathbf{v}$ has unit length. $|r| = \cos^2 \frac{1}{2} \theta + \sin^2 \frac{1}{2} \theta \mathbf{v} \cdot \mathbf{v} = 1$. $r^{-1} = (\cos \frac{1}{2} \theta, -\sin \frac{1}{2} \theta \mathbf{v})$. Let $p' = rpr^{-1}$. This quaternion can be shown to have the form $(0, p')$, and thus represents a point. It is the result of rotating $p$ by $\theta$ about the vector $\mathbf{v}$."
**Example**

- Consider a rotation about the $x$ axis
- $r = \cos \frac{1}{2}\theta + \sin \frac{1}{2}\theta \ (1, 0, 0)$
- $r^{-1} = \cos \frac{1}{2}\theta - \sin \frac{1}{2}\theta \ (1, 0, 0)$
- $p = (x, y, z)$
- $p' = rpr^{-1}$
- $rp = (r_0, r) (p_0, p) = (r_0 p_0 - r \cdot p, r_0 p + p_0 r + r \times p)$
  
  $= (-x \sin \frac{1}{2}\theta, \cos \frac{1}{2}\theta (x, y, z) + (0, -z \sin \frac{1}{2}\theta, y \sin \frac{1}{2}\theta)) = (-x \sin \frac{1}{2}\theta, (x \cos \frac{1}{2}\theta, y \cos \frac{1}{2}\theta - z \sin \frac{1}{2}\theta, z \cos \frac{1}{2}\theta + y \sin \frac{1}{2}\theta))$
Example (cont.)

- \( rpr^{-1} = (0, x, y \cos \theta - z \sin \theta, z \cos \theta + y \sin \theta) \)
- This is a point in space since the first term is 0
- \( p' = (x, y \cos \theta - z \sin \theta, y \sin \theta + z \cos \theta) \)
- This is the same result we got by using the rotation transformation matrix, but we actually used fewer operations
More information

- More information about quaternions can be found online at www.quaternions.com
Shears

- A shear is a transformation that produces distortion
- For example, x-direction shear distorts the horizontal position of a point by an amount that is proportional to its vertical component:

\[
\begin{bmatrix}
1 & h_x & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]
Shears in 3-D

The generalized shearing transformation matrix is

\[
\begin{bmatrix}
1 & h_{yx} & h_{zx} & 0 \\
h_{xy} & 1 & h_{zy} & 0 \\
h_{xz} & h_{yz} & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
Modeling what we see

- Observe the classroom we’re in
- Recognize there are many objects in the room
- The room can be modeled in many ways
- No matter which way we model the room, the modeling is done in world coordinates
View reference point (VRP)

- Imagine moving around the room with your head pointing in the same direction at all times
- Your viewing reference system remains the same
  - facing the same direction
  - peripheral vision is the same
  - see the same quantity of the room
  - see a different portion of the room
- Viewing reference with respect to the room changes
- View reference point specifies the position of the viewer with respect to the world coordinate system
- Specified in world coordinates
Viewplane normal (VPN)

- Now imagine standing still and swivelling your head left and right
- Your viewing reference changes but not your VRP (you’re stationary in the world coordinate system)
- The viewplane normal specifies the direction in which the viewer is looking
- Normally specified as a unit vector (n) in world coordinates
Up vector

- Need to specify a second direction to orient our viewing coordinate system
- When we look at a scene we maintain an intuitive sense of which direction is up
- We specify a unit vector, $\mathbf{v}$, in world coordinates to indicate the up direction
- Note that $\mathbf{v}$ is perpendicular to $\mathbf{n}$
The UVN system

- Have described a viewing reference system that is somewhat independent of the world coordinate system
- The viewplane is described by its normal ($\mathbf{n}$) and its up vector ($\mathbf{v}$)
- Need a third axis for a 3-D system
- Let $\mathbf{u} = \mathbf{v} \times \mathbf{n}$
- Origin of UVN system is center of viewplane window and eye is on positive $\mathbf{n}$ axis
OpenGL approach

- `gluLookAt (xeye, yeye, zeye, xcenter, ycenter, zcenter, xup, yup, zup);`
- Generates a transformation matrix that postmultiplies the current matrix
- Current matrix should be the model-view matrix
- Maps the eye point to the origin and the reference point to the negative z-axis
Example program

- `lookingAtCube.c` demonstrates viewing in OpenGL
Viewing coordinates

- Given $\mathbf{p} = (x, y, z)$ in the scene
- What are the viewing coordinates $(a, b, c)$ of $\mathbf{p}$?
- $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{n}$ are also given, as well as the VRP $\mathbf{r}$
- All we’ve got is a transformation from one system to the other
- What is this transformation?
1st step: translation

- Place the origin of the view coordinate system at the VRP, \( r \)
- This is nothing more than a translation by \( r \)
2nd step: rotation

- Once translated, UVN is just appropriate rotations of IJK
- Put the values that perform these rotations in a $3 \times 3$ matrix $M$
- From the fact that $u = Mi$ we conclude that the first column of $M$ must be the components of $u$
- Likewise, the second column of $M$ must be the components of $v$ and the third column the components of $n$
Combining the steps

\[
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix} = M \begin{bmatrix}
  a \\
  b \\
  c
\end{bmatrix} + \begin{bmatrix}
  r_x \\
  r_y \\
  r_z
\end{bmatrix}
\]

\[
\begin{bmatrix}
  a \\
  b \\
  c
\end{bmatrix} = M^{-1}(p - r)
\]
Further simplifications

- Recognize that $M$ is orthogonal, so its inverse is equal to its transpose

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = M^T(p - r)$$

- Since $M^T$ contains $u$, $v$, and $n$ as row vectors, $a$, $b$, and $c$ can be found with simple dot products
  - $a = u \cdot (p-r)$
  - $b = v \cdot (p-r)$
  - $c = n \cdot (p-r)$
Homogeneous form

- Putting this into homogeneous form means that $-\mathbf{M}^T \mathbf{r}$ has to be considered as the translation

- The composite transformation matrix is

$$
\begin{bmatrix}
  u_x & u_y & u_z & -u \cdot \mathbf{r} \\
  v_x & v_y & v_z & -v \cdot \mathbf{r} \\
  n_x & n_y & n_z & -n \cdot \mathbf{r} \\
  0 & 0 & 0 & 1
\end{bmatrix}
$$
Where we’re at

- This transformation takes us from world coordinates to view coordinates
- Can now perform an appropriate projection onto the view plane
Projections

- A projection takes a 3-dimensional image and displays it on a 2-dimensional surface
- Two main types
  - Orthographic
  - Perspective
Orthographic projection

- A parallel plane projection
- Most commonly used in engineering and architecture
- View scene along one of the principal axes
- Drop perpendiculars from the vertices to the viewing plane
- Effect is to make the coordinate for the axis you are viewing along 0
Problems with orthographic projection

- You lose information in each projection
- To show all information in a scene requires six different projections
  - Top
  - Rear
  - Left
  - Bottom
  - Front
  - Right
Projections in OpenGL

- Image must reside in negative z space
- All z values specified to projection commands are usually positive - OpenGL negates them automatically
- However, if the near or far plane should be behind the viewer, use a negative number
Orthogonal projections in OpenGL

- `glOrtho (left, right, bottom, top, near, far);`
- `gluOrtho2D (left, right, bottom, top);`  
  - equivalent to `glOrtho` with `near=-1` and `far=1`
More on OpenGL projections

- *near* must never be set to 0 for 3-D
- Depth buffer precision is affected by the ratio of *far* to *near*
- In the projection pipeline the projection matrix is applied after the model-view matrix
Perspective projection

- Provides depth cueing information
- Introduces some distortion into the image, but this distortion is good
- Perspective projection is really a perspective transformation (of a 3-D object into a 3-D object) followed by an orthographic projection of the transformed object
Geometry of perspective

- Let eye be placed at the origin and look toward the negative $z$ axis.
- Consider a projector from the eye to a point $P(x, y, z)$ on the object which passes through the view plane $z = d$, where $d$ is negative.
- The equation for this projector is $p(t) = (x, y, z)t + (0, 0, 0)(1-t)$. 
Geometry of perspective (cont.)

- The time at which this hits the view plane is found to be $t = d/z$
- Therefore, the point at which the projection hits the view plane is $P' = (x/(z/d), y/(z/d), d)$
Use of the homogeneous coordinate

- Note that the $x$ and $y$ coordinates are divided by $z/d$
- Recall the 1 in the homogeneous coordinates of a point
- We let the homogenous coordinate indicate the division:
  \[(x/(z/d), y/(z/d), d, 1) = (x, y, z, z/d)\]
Perspective transformation matrix

Putting this division into matrix form results in the following transformation matrix:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0 \\
\end{bmatrix}
\]

Note that the last column is all zeroes, making the matrix singular.
Perspective projections in OpenGL

- `glFrustum (left, right, bottom, top, near, far);
- `gluPerspective (fovy, aspect, near, far);
  - `fovy is field of view in y axis
  - `aspect is aspect ratio (width/height)
Example program

- viewRotatingCube.cpp demonstrates perspective projections in OpenGL
Lines

- Consider the parametric form for a line, $p(t) = q + rt$, where $q$ is the starting point and $r$ is the direction.

- Plugging this into the perspective transformation matrix yields:

$$
\begin{bmatrix}
1 & 0 & 0 & 0 & q_x + r_xt \\
0 & 1 & 0 & 0 & q_y + r_yt \\
0 & 0 & 1 & 0 & q_z + r_zt \\
0 & 0 & 1/d & 0 & 1 \\
0 & 0 & 0 & 1 & d
\end{bmatrix}
= \begin{bmatrix}
q_x + r_xt \\
q_y + r_yt \\
q_z + r_zt \\
1 \\
d
\end{bmatrix}
$$
What happens as $t \to \infty$?

- Consider the $x$ coordinate of the transformed point:

  \[
  \frac{q_x + r_x t}{q_z + r_z t} = \frac{dq_x + dr_x t}{q_z + r_z t} \tag{d}
  \]

- Let the value of $t$ approach $\infty$:

  \[
  \frac{dr_x t}{r_z t} = \frac{dr_x}{r_z}
  \]

- Note that this is independent of $q$ and $t$
Vanishing points

- All lines which have the same direction \((r)\) will meet at this point.
- This means parallel lines will meet.
- This point is known as the **vanishing point**.
Types of perspective projections

- **One-point perspective**
  - Viewing along a principal axis
  - One vanishing point

- **Two-point perspective**
  - Rotating viewpoint around an axis but keeping it in a principal plane
  - Two vanishing points

- **Three-point perspective**
  - Rotating viewpoint off a principal plane
  - Three vanishing points