CS 4204 Computer Graphics

3D Viewing
Adapted from notes by Yong Cao
Virginia Tech
The Camera and Perspective Projection

- **Camera properties:**
  - Eye (or view reference point VRP) at some point in space
  - View volume: a portion of a pyramid, whose apex is at the eye
  - View plane normal, or VPN
  - Field of view $\theta$
  - Near and far clipping planes
  - Viewplane and its aspect ratio
The Camera and Perspective Projection
Setting the View Volume

• The default camera position has the eye at the origin and the VPN aligned with the z-axis.

• The programmer defines a look point as a point of particular interest in the scene, and together the two points eye and look define the VPN as eye – look.

• This is later normalized to become the vector \( \mathbf{n} \), which is central in specifying the camera properly. (VPN points from look to eye.)
Setting the View Volume (2)
Setting the View Volume (3)

• *To view a scene, we move the camera and aim it in a particular direction.*

• *To do this, perform a rotation and a translation, which become part of the modelview matrix.*

```c
glMatrixMode(GL_MODELVIEW); // make the modelview matrix current
glLoadIdentity(); // start with a unit matrix
gluLookAt(eye.x, eye.y, eye.z, look.x, look.y, look.z, up.x, up.y, up.z);
```
Setting the View Volume (4)

• As before, this moves the camera so its eye resides at point eye, and it “looks” towards the point of interest, look.

• The “upward” direction is generally suggested by the vector up, which is most often set simply to (0, 1, 0).
## Camera with Arbitrary Orientation and Position

- A camera can have any position and orientation in the scene.
- Imagine a transformation that picks up the camera and moves it somewhere in space, then rotates it around to aim it as desired.
- To do this we need a coordinate system attached to the camera: $u$, $v$, and $n$. 

<table>
<thead>
<tr>
<th>Camera with Arbitrary Orientation and Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>- A camera can have any position and orientation in the scene.</td>
</tr>
<tr>
<td>- Imagine a transformation that picks up the camera and moves it somewhere in space, then rotates it around to aim it as desired.</td>
</tr>
<tr>
<td>- To do this we need a coordinate system attached to the camera: $u$, $v$, and $n$.</td>
</tr>
</tbody>
</table>
Camera with Arbitrary Orientation and Position (2)

\( v \) points vertically upward, \( n \) away from the view volume, and \( u \) at right angles to both \( n \) and \( v \). The camera looks toward \(-n\). All are normalized.
gluLookAt and the Camera Coordinate System

*gluLookAt* takes the points *eye* and *look*, and the vector *up*

*n must be parallel to eye - look*, so it sets \( n = \text{eye} - \text{look} \)

*u points "off to the side", so it makes u perpendicular to both n and up*: \( u = \text{up} \times n \)

*\( v \) must be perpendicular to n and u, so it lets* \( v = n \times u \)
gluLookAt and the Camera Coordinate System (2)

Effect of $\text{gluLookAt (Demo)}$
The view matrix \( V \) created by \texttt{gluLookAt} is

\[
V = \begin{pmatrix}
u_x & u_y & u_z & d_x \\
v_x & v_y & v_z & d_y \\
n_x & n_y & n_z & d_z \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

where \( d_x = -\text{eye} \cdot u \), \( d_y = -\text{eye} \cdot v \), \( d_z = -\text{eye} \cdot n \)

\( V \) is postmultiplied by \( M \) to form the modelview matrix \( VM \).
Perspective Projections of 3-D Objects

The graphics pipeline: vertices start in world coordinates; after VM, in eye coordinates, after P, in clip coordinates; after perspective division, in normalized device coordinates; after V, in screen coordinates.
Mathematics of a basic perspective projection

Similar triangles

\[
\begin{align*}
x' &= \frac{x}{d} = \frac{x}{z}
\end{align*}
\]

\[
\begin{align*}
y' &= \frac{y}{d} = \frac{y}{z}
\end{align*}
\]

\[
\begin{align*}
x' &= \frac{x}{(z/d)}
\end{align*}
\]

\[
\begin{align*}
y' &= \frac{y}{(z/d)}
\end{align*}
\]

\[
\begin{align*}
z' &= d
\end{align*}
\]
One-Point Projection in matrix form

Center of Projection at the origin with viewplane parallel to the x-y plane a distance d from the origin.

\[ x_{\text{projected}} = \frac{x}{z/d} \]
\[ y_{\text{projected}} = \frac{y}{z/d} \]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1/d & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
z/d
\end{bmatrix}
=
\begin{bmatrix}
x/(z/d) \\
y/(z/d) \\
d \\
z/d
\end{bmatrix}
\]

**Points plotted are x/w, y/w where w = z/d**
A 3D viewing example
Incorporating Perspective in the Graphics Pipeline

• *We need to add depth information (destroyed by projection).*

• *Depth information tells which surfaces are in front of other surfaces, for hidden surface removal.*
Incorporating Perspective in the Graphics Pipeline (2)

Instead of Euclidean distance, we use a pseudodepth, $-1 \leq P_z' \leq 1$ for $-N > z > -F$. This quantity is faster to compute than the Euclidean distance.

$P_z'$ increases (becomes more positive) as $P_z$ decreases (becomes more negative, moves further away).
Illustration of Pseudo-depth Values
Pseudodepth values bunch together as $-P_z$ gets closer to $F$, causing difficulties for hidden surface removal.

When $N$ is much smaller than $F$, as it normally will be, pseudodepth can be approximated by

$$\text{pseudodepth} \approx 1 + \frac{2N}{P_z}$$
Using homogeneous coordinates allows us to capture perspective using a matrix multiplication. To make it work, we must always divide through by the fourth (w) component, a step which is called perspective division.
A matrix that has values other than (0,0,0,1) for its fourth row does not perform an affine transformation. It performs a more general class of transformation called a perspective transformation.

It is a transformation, not a projection. A projection reduces the dimensionality of a point, to a 3-tuple or a 2-tuple, whereas a perspective transformation takes a 4-tuple and produces a 4-tuple.
Perspective Transformation and Homogeneous Coordinates (3)

Where does the projection part come into play?

- The first two components of this point are used for drawing: to locate in screen coordinates the position of the point to be drawn.

- The third component is used for depth testing.

As far as locating the point on the screen is concerned, ignoring the third component is equivalent to replacing it by 0; this is the projection part (as in orthographic projection, Ch. 5).
Perspective Transformation and Homogeneous Coordinates (4)

\[(\text{perspective projection}) = (\text{perspective transformation}) + (\text{orthographic projection})\].

- OpenGL does the transformation step separately from the projection step.
- It inserts clipping, perspective division, and one additional mapping between them.
Perspective Transformation and Homogeneous Coordinates (5)

• When we wish to display a mesh model we must send thousands or even millions of vertices down the graphics pipeline.

• Clearly it will be much faster if we can subject each vertex to a single matrix multiplication rather than to a sequence of matrix multiplications.

• This is what OpenGL does: it multiplies all of the required matrices into a single matrix once and then multiplies each vertex by this combined matrix.
Geometry of Perspective Transformation

• The perspective transformation alters 3D point P into another 3D point, to prepare it for projection. It is useful to think of it as causing a warping of 3D space and to see how it warps one shape into another.

• Very importantly, it preserves straightness and flatness, so lines transform into lines, planes into planes, and polygonal faces into other polygonal faces.

• It also preserves in-between-ness, so if point a is inside an object, the transformed point will also be inside the transformed object.

• Our choice of a suitable pseudodepth function was guided by the need to preserve these properties.
Geometry of Perspective Transformation (2)

How does it transform the camera view volume?

We must clip to this volume.
### Geometry of Perspective Transformation (3)

- **The near plane** $W$ at $z = -N$ maps into the plane $W'$ at $z = -1$, and the far plane maps to the plane at $z = +1$.

- **The top wall** $T$ is tilted into the horizontal plane $T'$ so that it is parallel to the $z$-axis.

- **The bottom wall** $S$ becomes the horizontal $S'$, and the two side walls become parallel to the $z$-axis.

- **The camera's view volume is transformed into a parallelepiped.**
The transformation also warps objects into new shapes.

The perspective transformation warps objects so that, when viewed with an orthographic projection, they appear the same as the original objects do when viewed with a perspective projection.
Geometry of Perspective Transformation (5)

• OpenGL composes the perspective transformation with another mapping that scales and translates this parallelepiped into the canonical view volume, a cube that extends from -1 to 1 in each dimension.

• Because this scales things differently in the x- and y-dimensions, it introduces some distortion, but the distortion will be eliminated in the final viewport transformation.
Perspective Projection Matrix used by OpenGL

\[
R = \begin{bmatrix}
\frac{2N}{right - left} & 0 & \frac{right + left}{right - left} & 0 \\
0 & \frac{2N}{top - bottom} & \frac{top + bottom}{top - bottom} & 0 \\
0 & 0 & \frac{-(F + N)}{F - N} & -2FN \\
0 & 0 & \frac{F - N}{F - N} & 0
\end{bmatrix}
\]
Perspective Projection Matrix (2)

`gluPerspective(viewAngle, aspect, N, F)` is usually used instead of `glFrustrum(left, right, bottom, top, near, far)`, as its parameters are more intuitive.

`gluPerspective()` sets up the same matrix, after computing values for top, bottom, etc. using

\[
\begin{align*}
top &= N \tan\left(\frac{\pi}{180} \frac{viewAngle}{2}\right) \\
bott &= -top, \quad right = top \times aspect, \quad and \quad left = -right.
\end{align*}
\]
The Graphics Pipeline in OpenGL
Following clipping, perspective division is finally done and the 3-tuple \((x, y, z)\) is passed through the viewport transformation.

The perspective transformation squashes the scene into the canonical cube. If the aspect ratio of the camera’s view volume (that is, the aspect ratio of the window on the near plane) is not 1.0, there is obvious distortion introduced.

But the viewport transformation can undo this distortion by mapping a square into a viewport of aspect ratio 1.5. We normally set the aspect ratio of the viewport to be the same as that of the view volume.
Distortion and Its Removal

`glViewport(x, y, wid, ht)` specifies that the viewport will have lower left corner \((x,y)\) in screen coordinates and will be \(wid\) pixels wide and \(ht\) pixels high. It thus specifies a viewport with aspect ratio \(wid/ht\).

The viewport transformation also maps pseudodepth from the range -1 to 1 into the range 0 to 1.
Steps in the Pipeline: Each Vertex $P$ Undergoes Operations Below

- $P$ is extended to a homogeneous 4-tuple by appending a 1, and this 4-tuple is multiplied by the modelView matrix, producing a 4-tuple giving the position in eye coordinates.

- The point is then multiplied by the projection matrix, producing a 4-tuple in clip coordinates.

- The edge having this point as an endpoint is clipped.

- Perspective division is performed, returning a 3-tuple.

- The viewport transformation multiplies the 3-tuple by a matrix; the result $(s_x, s_y, d_z)$ is used for drawing and depth calculations. $(s_x, s_y)$ is the point in screen coordinates to be displayed; $d_z$ is a measure of the depth of the original point from the eye of the camera.