Taking a picture with a camera

- Coordinates: Local, World, Viewing
- Rendering pipeline
- ModelView
  - Matrix operations on models
- World coordinates to Viewing coordinates
  - Matrix operations (models or cameras)
- Projection with a camera

Positioning the Camera

- By default, the camera is placed at the origin pointing in the negative z-axis.

```plaintext
OpenGL Look-At Function

gluLookAt(eyex, eyey, eyez, atx, aty, atz, upx, upy, upz)
```

- View-reference point (VRP)
- View-plane normal (VPN)
- View-up vector (VUP)

View Transformation Matrix

- View Matrix = [ ux uy uz -u.X ]
- [ vx vy vz -v.X ]
- [ nx ny nz -n.X ]

Chalk Talk
Projection in 3D

- Planar Geometric Projections
- Parallel Orthographic Projections
- Perspective Projections
- Projections in OpenGL
- Clipping

Planar Geometric Projections

- Maps points from camera coordinate system to the screen (image plane of the virtual camera).

Parallel Orthographic Projection

- Preserves X and Y coordinates.
- Preserves both distances and angles.

Perspective Projection

- Only preserves parallel lines that are parallel to the image plane.
- Line segments are shorten by distance.
Perspective Projection

- \( z_p = d \)
- \( x_p = \frac{(x \cdot d)}{z} \)

Viewing in 3D

- Planar Geometric Projections
- Parallel Orthographic Projections
- Perspective Projections
- Projections in OpenGL
  - Positioning of the Camera
  - Define the view volume

Defining the Perspective View Volume

```glFrustum(left, right, bottom, top, near, far)```
Defining the Perspective View Volume

\[ \text{gluPerspective} (\text{fovy}, \text{aspect}, \text{near}, \text{far}) \]

Defining the Parallel View Volume

\[ \text{glOrtho} (\text{xmin}, \text{xmax}, \text{ymin}, \text{ymax}, \text{near}, \text{far}) \]

Clipping

2D Line Clipping

- Cohen-Sutherland algorithm

<table>
<thead>
<tr>
<th>Bit 1: left</th>
<th>Bit 2: right</th>
<th>Bit 3: below</th>
<th>Bit 4: above</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>1000</td>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>0001</td>
<td>0000</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>0101</td>
<td>0100</td>
<td>0110</td>
<td></td>
</tr>
</tbody>
</table>

Cohen-Sutherland algorithm

- The space is divided into nine regions.
- Each region is assigned a unique 4-bit binary number: outcode

<table>
<thead>
<tr>
<th>1001</th>
<th>1000</th>
<th>1010</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>0000</td>
<td>0010</td>
</tr>
<tr>
<td>0101</td>
<td>0100</td>
<td>0110</td>
</tr>
</tbody>
</table>
Cohen-Sutherland algorithm

- Each line has two endpoints with two outcodes:
  \(o_1=\text{outcode} (x_1, y_1)\) and \(o_2=\text{outcode} (x_2, y_2)\).
- We can decide based on these two outcodes.

Four cases:

1. \(o_1 = o_2 = 0\). Both endpoints are inside the window. Such as AB.
2. \(o_1 \neq 0, o_2 = 0\), or vice versa. One endpoint is inside, the other is outside. Such as CD.
3. \(o_1 \neq 0, o_2 \neq 0\). Whether or not the two endpoints lie on the same outside of the window. Such as EF and GH.
4. \(o_1 = 0, o_2 \neq 0\). Both endpoints are outside, but there are on the outside of the different edges of the window. Such as IJ.