I. Recall the viewing pipeline

Model → World → Viewing → Normalized → Device

A. Viewing is concerned w/ last 3 portions of the pipeline. Moreover, it concerns itself w/ which primitives are visible and therefore should be rendered.

B. Lesson: more primitives = larger in framerate. So if we call line alg or fill alg on something we can't see from our viewing transform this slows our app significantly.

C. Problems: (1) How to do transforms (2) How to decide what is visible.

II. Two-dimensional Viewing (Chapter 8)

A. Two-dimensional viewing terms

i. Clipping = window - section of 3D world (scene) selected for display

ii. Clipping - procedurally eliminate portions of a picture that are outside a specified region of space. Usually a rectangle is considered as the region

iii. Viewport - portion in window to display contents of clipping window

iv. Clipping says "what" we see, "view" states "where" we see it.

v. We will consider rectangles for clipping/viewing

\[
\text{Clipping} \Rightarrow (X_{min}, Y_{min}, X_{max}, Y_{max}) \quad \text{Viewport} \Rightarrow (X_{min}, Y_{min}, X_{max}, Y_{max})
\]

vi. In general - clipping/viewports could be any shape and multiple viewports are allowed in applications, e.g. CAD.

*Question: How could we support a region that is not axis aligned? Also is how a menu could be made.

B. Normalization and Viewport Transformations

i. Clipping window to normalized

\[
\begin{align*}
\text{Clipping window} & \quad \Rightarrow \quad \text{Normalized viewport} \\
\text{Viewport in window} & \quad \Rightarrow \quad \text{Screen viewport}
\end{align*}
\]
To transform any point \((x, y)\) from the window to the viewport \((x', y')\), the following must hold:

\[
\frac{x' - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}, \quad \text{and} \quad \frac{y' - y_{\text{min}}}{y_{\text{max}} - y_{\text{min}}} = \frac{y - y_{\text{min}}}{y_{\text{max}} - y_{\text{min}}}
\]

Solving for \((x', y')\), we get

\[
x' = s_x x + t_x \quad y' = s_y y + t_y
\]

where

\[
s_x = \frac{x_{\text{max}} - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}, \quad t_x = \frac{x_{\text{max}} x_{\text{min}} - x_{\text{min}} x_{\text{max}}}{x_{\text{max}} - x_{\text{min}}},
\]

\[
s_y = \frac{y_{\text{max}} - y_{\text{min}}}{y_{\text{max}} - y_{\text{min}}}, \quad t_y = \frac{y_{\text{max}} y_{\text{min}} - y_{\text{min}} y_{\text{max}}}{y_{\text{max}} - y_{\text{min}}}.
\]

So (1) apply scale \((s_x, s_y)\), fixed point \((x_{\text{min}}, y_{\text{min}})\), (2) Translate \((x_{\text{min}}, y_{\text{min}})\) to \((0, 0)\)

\[
M = TS = \begin{bmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}
\]

Note: Relative proportions of objects are only maintained if the aspect ratio of the viewport is the same as the clipping window.

1. Clipping window to normalized square (second approach)

- Essentially apply transformation above twice. First window as \((-1, -1), (1, 1))\)
- Second window as \((-1, -1), (1, 1))\), why: simplifies computations.

6. Point Clipping *Ask class how to do this*

\[x_{\text{min}} \leq x \leq x_{\text{max}}, \quad y_{\text{min}} \leq y \leq y_{\text{max}} \quad \text{must be satisfied, else throw away.}\]
D. Line Clipping

1. First Idea: Compute intersections with lines w/ clipping region.

2. Second Idea: Determine which lines are entirely inside or outside. Only compute intersections where necessary. How? Apply point test to endpoints, if both are inside then the line is inside. Otherwise it is harder to determine if completely outside. If both endpoints are outside the same boundary it is outside.

\[ x = x_0 + u \Delta x \quad y = y_0 + u \Delta y \quad 0 \leq u \leq 1 \]

Substitute boundary into one equation and solve for u. If u is outside [0,1] the line does not intersect that barrier, otherwise part of the line is inside. Continue in all others until clip boundary is determined.

ii. Cohen-Sutherland Line Clipping

a. First every endpoint is assigned a 4-bit region code. If 1 bit is outside 0 is inside

<table>
<thead>
<tr>
<th>Bit Pattern</th>
<th>Region Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1001</td>
<td>Top left</td>
</tr>
<tr>
<td>0100</td>
<td>Bottom right</td>
</tr>
<tr>
<td>0010</td>
<td>Right</td>
</tr>
<tr>
<td>0001</td>
<td>Left</td>
</tr>
<tr>
<td>1000</td>
<td>Inside</td>
</tr>
</tbody>
</table>

b. Inside-outside check - if both endpoints are in 0000 it is completely inside, if any bit is 1 in both it is completely outside. Can do through bit operations.

Logical or results in 0000 \(\rightarrow\) inside

Logical and results in not(0000) \(\rightarrow\) true \(\rightarrow\) outside

C. Clipping process to check boundary last time if bit codes are different on endpoints we compute intersection point. Throw away outside portion, repeat for all boundary segments.

D. Faster algo exist but inside scope of lecture.
E. Polygon-Fill Clipping

i. Cannot simply close edges because result would not be closed polyline.

ii. Can progressively clip polygon and determine if it is entirely inside/outside.

iii. Basic idea - Create new vertex list at each clipping boundary. Pass new vertex list to next clipper.

iv. Sutherland-Hodgemann Polygon Clipping
   a. Basic Idea - Send pair of endpoints for each segment through clippers. Send successive pair of coordinates to next clipper. 4 cases arise based on inside/outside of endpoints.

b. Rules to pass vertices on:
   1. If first vertex is outside and second is inside, then send both intersection point and second vertex.
   2. If both are inside, then send the second.
   3. If first is inside and second is outside, then only the intersection point on.
   4. If both are outside, send both vertices on.

iv. Example

Input edge $\rightarrow$ left $\rightarrow$ right $\rightarrow$ Bottom $\rightarrow$ top

$(1,2)$: in/in $\rightarrow$ \{2\}

$(2,3)$: out/in $\rightarrow$ \{2\}

$(3,4)$: out/in $\rightarrow$ \{2\}

$(4,5)$: out/in $\rightarrow$ \{2\}

$(5,6)$: out/in $\rightarrow$ \{2\}

$(6,7)$: out/in $\rightarrow$ \{2\}

$(7,8)$: out/in $\rightarrow$ \{2\}

$(8,9)$: out/in $\rightarrow$ \{2\}

$(9,10)$: out/in $\rightarrow$ \{2\}

$(10,11)$: out/in $\rightarrow$ \{2\}

$(11,12)$: out/in $\rightarrow$ \{2\}
V. Weiler-Atkerson Polygon Clipping

a. Basic idea - trace polygon boundary to produce clipped fill region
b. In a counter-clockwise order
1. Process until input is found. Compute intersection point
2. Follow window boundaries until another intersection point is met. Continue until
   a previously processed point is met.
3. Form vertex list
4. Return to exit point, continue processing edges until all regions found

Example

F. Can also clip curves, text, other regions, etc.

III. 3D Viewing (chapter 16)

A. Viewing concepts
i. Camera - we first need a coordinate reference frame for viewing. It defines a view plane
ii. We have choice in projections
   a. Parallel projection - used in CAD, parallel lines stay parallel
   b. Perspective projection - lines converge at distance, mimics reality

iiia. Depth cueing - need to know what is front/back of an object. Closer objects appear brighter.
    or more intense
iii. Visible object detection - depth relationships of an object being in front (Ch. 16)
iv. Surface rendering - lighting/detaling surfaces (Ch. 17)

v. More such as stereoscopic viewing for 3D glasses
B. Three-dimensional viewing pipeline

\[ MC \rightarrow \text{Transforms} \rightarrow WC \rightarrow \text{Viewing Transform} \rightarrow \text{View} \rightarrow \text{Coordinates} \]

\[ \rightarrow \text{Projection} \rightarrow \text{Proj} \rightarrow \text{Normalized} \rightarrow \text{proj} \rightarrow \text{Coordinate} \rightarrow \text{Viewport} \rightarrow \text{Device} \rightarrow \text{Coordinates} \]

\[ \text{and clipping} \]

\[ \text{with clipping planes surrounding a viewing volume} \]

C. Three-dimensional viewing-coordinate parameters

i. Pick a world position for viewing \( P_0 \), called view point or eye position.
ii. Pick the view plane normal direction \( \vec{N} \). Defines Z-axis of view coordinates.

\[ \begin{align*}
\vec{N} &= \frac{\vec{Z}}{|\vec{Z}|} = (x_N, y_N, z_N) \\
\vec{O} &= \frac{\vec{V} \times \vec{Z}}{|\vec{V} \times \vec{Z}|} = (x_V, y_V, z_V) \\
\vec{V} &= \vec{N} \times \vec{O} = (x_V, y_V, z_V)
\end{align*} \]

Zp defines how far in the Z-axis the plane is.

iii. Pick an up vector \( \vec{U} \) to define Y-axis of view coordinates (not parallel to \( \vec{N} \)).
iv. UVN reference frame (get full axis information)

\[ \begin{align*}
\vec{N} &= \frac{\vec{Z}}{|\vec{Z}|} = (x_N, y_N, z_N) \\
\vec{O} &= \frac{\vec{V} \times \vec{Z}}{|\vec{V} \times \vec{Z}|} = (x_V, y_V, z_V) \\
\vec{V} &= \vec{N} \times \vec{O} = (x_V, y_V, z_V)
\end{align*} \]

Z-axis \( x \)-axis \( y \)-axis

D. Transformation from world coordinates to viewing coordinates

i. Translate view origin to world origin.
ii. Rotate to align coordinate frame to world frame.

\[ T = \begin{bmatrix}
1 & 0 & 0 & x_0 \\
0 & 1 & 0 & y_0 \\
0 & 0 & 1 & z_0 \\
0 & 0 & 0 & 1
\end{bmatrix} \quad R = \begin{bmatrix}
v_1 & v_2 & v_3 & 0 \\
v_4 & v_5 & v_6 & 0 \\
v_7 & v_8 & v_9 & 0 \\
v_{10} & v_{11} & v_{12} & 1
\end{bmatrix} \quad M = RT = \begin{bmatrix}
v_1 & v_2 & v_3 & x_0 \\
v_4 & v_5 & v_6 & y_0 \\
v_7 & v_8 & v_9 & z_0 \\
v_{10} & v_{11} & v_{12} & 1
\end{bmatrix} \]
6. Orthogonal Projections - transform of object along parallel lines to viewplane

(Recall we have oriented our camera)

i. Axometric - display more than one plane
   Isometric - axometric such that viewplane intersects each coordinate axis at the same distance from the origin

ii. Orthogonal Projection Coordinates
   a. Direction or projection is parallel to view axis, so it is trivial
   b. Projection
      \[ \begin{array}{ccc}
      X' & = & X \\
      Y' & = & Y \\
      Z' & = & Z \\
      \end{array} \]
   c. Orthogonal projection view volume
      a. sides of clipping window define finite region of clipping

   \[ \text{Clipping Window} \]

6. We specify near/far clipping planes to limit output. Since direction is negative view axis, \( z_{\text{near}} < z_{\text{far}} \)

7. Shape is rectangular parallelepiped

8. Normalization transformation - transfer to symmetric cube \((1,1,1)\) for clipping:

   \[ M_{\text{normalization}} = \begin{bmatrix}
   2 & 0 & 0 & 0 \\
   0 & 2 & 0 & 0 \\
   0 & 0 & 2 & 0 \\
   0 & 0 & 0 & 1 \\
   \end{bmatrix} \]

   \[ \text{Note: } z ' \text{'s negative to flip to left-handed coordinates. Otherwise: } z_{\text{near}} \text{'s need to be} \]

   multiply on right of model-view transformation, then can apply to object all at once.
F. Oblique Parallel Projection - parallel lines in a direction relative to view plane

\[ L - \text{length of viewport line} \]
\[ \alpha - \text{intersection angle} \]
\[ \phi - \text{angle of line from horizontal} \]

For combined Top, Front, Side view

\[ \tan \alpha = \frac{2y_p - L}{L} \]

\[ L = \frac{2L_p - 2}{\tan \alpha} \]

So \[ x_p = x + L \cos \beta \]
\[ y_p = y + L \sin \beta \]

3 represents Shearing

\[ \alpha = 90^\circ \]

\[ \beta = \frac{\pi}{2} \]

\[ \frac{\pi}{2} \text{ is usually } 30^\circ \text{ or } 45^\circ \]

\[ \alpha = 45^\circ \text{ is a Cavalier projection} \]

\[ \alpha = 30^\circ \text{ is a Cabinet projection} \]

V. Transform

\[ \begin{bmatrix}
1 - \frac{y_p}{V_p} & \frac{z_p}{V_p} & 0 \\
0 & 1 & \frac{z_p}{V_p} \\
0 & 0 & 1
\end{bmatrix} \]

\[ \binom{0}{0}{0} \]

\[ \text{Slope: Oblique parallelepiped} \]

VI. Same mapping to normalized cube

\[ \text{Mobilization} = \text{Montgomery Mobilize} \]
G. Perspective Projection - parallel lines converge, more realistic

- Distant objects appear smaller

\[
\begin{align*}
\text{Perspective projection:} \\
\text{(Projective reference point)} \\
\text{(Convergence point)} \\
(x_{pp}, y_{pp}, z_{pp})
\end{align*}
\]

- Perspective projection variants

- \(z_{pp}\) is 1 or view plane

- So from perspective form of line: \(0 \leq u \leq 1\)

\[
\begin{align*}
\frac{u}{z_{pp}} &= \frac{z_{pp} - 2z}{z_{pp}} \\
x_p &= u \left( \frac{3z_{pp} - 2z_{pp}}{z_{pp}} \right) + x_{pp} \left( \frac{2z_{pp} - z}{z_{pp}} \right) \\
y_p &= y \left( \frac{3z_{pp} - 2z_{pp}}{z_{pp}} \right) + y_{pp} \left( \frac{2z_{pp} - z}{z_{pp}} \right)
\end{align*}
\]

- Problem: denominators are functions of \(z\) coordinate

To help, we can restrict reference point, e.g. by making it the coordinate origin.

Note: if reference point is in front of new plane, image is inverted.

H. View volume

- Burning planes are not parallel anymore, so we have an infinite rectangular pyramid.

I. Near-foe planes we have a frustum

J. Perspective-Projective Transformation Matrix

- Because of \(z\) coordinate dependency, we can exploit homogenous coordinates

\[
x_p = \frac{u}{1} \\
y_p = \frac{y}{1} \\
z_p = z_{pp} - z
\]

- So \(x_p = x(3z_{pp} - 2z_{pp}) + x_{pp} (2z_{pp} - z) \) \(y_p = y(3z_{pp} - 2z_{pp}) + y_{pp} (2z_{pp} - z)\)

- So \(P = M_{pp} P\) many choices for \(M_{pp}\) but we need one to preserve \(z\) information.

\[
M_{pp} = \begin{bmatrix}
3z_{pp} & -x_{pp} & x_{pp}z_{pp} \\
2z_{pp} & -y_{pp} & y_{pp}z_{pp} \\
0 & 0 & 1
\end{bmatrix}
\]

- Converts scene to homogenous parallel projection coordinates, i.e. on oblique projection. So we have 2 cases: symmetric projection, i.e. orthogonal, or non, i.e. oblique.
iv. Symmetric perspective projection frustrum - line through center of

\[ \text{is perpendicular to} \quad \text{cw} \]

\[ \text{View volume transformation} \quad \rightarrow \]

\[ \text{Orthonormal parallel projection} \]

- In this case, CW coordinates can be expressed in window coordinates.
- We can specify by field-of-view angle + aspect ratio.
- CW height is: \( 2 \times (z_{	ext{far}} - z_{	ext{up}}) \tan \left( \frac{\theta}{2} \right) \), and diagonal elements of matrix are updated to \( z_{	ext{far}} - z_{	ext{up}} = \text{height} \times \cot \left( \frac{\theta}{2} \right) \).

v. After apply \( \text{perspective transformation} \)

\[ \text{Matrix of perspective transformation} = \begin{bmatrix}
    -\frac{2 \text{near}}{\text{field-of-view} \times \text{aspect}} & 0 & \frac{\text{near} + \text{far}}{\text{field-of-view} \times \text{aspect}} & 0 \\
    0 & -\frac{\text{field-of-view}}{\text{far}} & 0 & 0 \\
    0 & 0 & \text{far} & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} \]

vi. After \( \text{perspective projection}, \) divide coordinates by homogeneous parameter. Set \( \text{frustum} \) transformed to \( \text{orthogonal parallel projection} \).

- So we need to scale \( x, y \) coordinates + determine \( z_{2}, z_{2} \).

\( \text{General perspective matrix becomes} \)

\[ \begin{bmatrix}
    -2 \text{near} & 0 & \text{near} + \text{far} & 0 \\
    0 & -\text{field-of-view} & 0 & 0 \\
    0 & 0 & \text{far} & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} \]

and

\[ \begin{bmatrix}
    \text{cot} \left( \frac{\theta}{2} \right) & 0 & 0 & 0 \\
    0 & \text{field-of-view} & 0 & 0 \\
    0 & 0 & 1 & 0 \\
    0 & 0 & 0 & 1
\end{bmatrix} \]

Multiply this on left of \( \text{view transformation} \), then apply clipping.
H. Viewport to twic3 transform

\[
M_{\text{viewport to twic3}} = \begin{bmatrix}
\frac{x_{\text{min}} - x_{\text{max}}}{2} & 0 & 0 & \frac{x_{\text{max}} + x_{\text{min}}}{2} \\
0 & \frac{y_{\text{min}} - y_{\text{max}}}{2} & 0 & \frac{y_{\text{max}} + y_{\text{min}}}{2} \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Z coordinate is mapped to depth buffer.