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. Lessons: More primitives = big in frame rate. 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 viewable.

II. Two-dimensional Viewing (Chapter 8)

A. Two-dimensional viewing terms

1. Clipping window - section of 3D world (scene) selected for display
2. Clipping - procedurally eliminating portions of a picture that are "outside a specified region of space. Usually a rectangle is considered as the region.
3. Viewport - portion in window to display contents of clipping window
4. Clipping says what we see, "here" states "where" we see it.
5. We will consider rectangles for clipping/viewing

\[
\text{Clipping} : (x_{	ext{min}}, y_{	ext{min}}, x_{	ext{max}}, y_{	ext{max}}) \quad \text{Viewport} : (x_{	ext{Canvas Min}}, y_{	ext{Canvas Min}}, x_{	ext{Canvas Max}}, y_{	ext{Canvas Max}})
\]

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

*Question: How could we support a region that is not axes aligned?

Also is how a menu could be made.

B. Normalization and Viewport Transformations

1. Clipping window to viewport transformation

\[
\begin{array}{c|c|c|c}
\text{Clipping window} & \Rightarrow & \text{Normalized viewport} & \Rightarrow & \text{Screen viewport}
\end{array}
\]
To transform any point \((x_u, y_u)\) from the window to viewport \((x_v, y_v)\), the following must hold:

\[
\frac{x_u - x_{umin}}{x_{umax} - x_{umin}} = \frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}} \quad \text{and} \quad \frac{y_u - y_{umin}}{y_{umax} - y_{umin}} = \frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}}
\]

Solving for \((x_v, y_v)\) we get:

\[
x_v = s_x x_u + t_x \quad y_v = s_y y_u + t_y \quad \text{where}
\]

\[
s_x = \frac{x_{umax} - x_{umin}}{x_{umax} - x_{umin}} \quad t_x = \frac{x_{umax} x_{vmin} - x_{vmin} x_{umin}}{x_{umax} - x_{umin}}
\]

\[
s_y = \frac{y_{umax} - y_{umin}}{y_{umax} - y_{umin}} \quad t_y = \frac{y_{vmax} y_{umin} - y_{umin} y_{vmax}}{y_{umax} - y_{umin}}
\]

1. Apply scale \(S\) w/ fixed point \((x_{umin}, y_{umin})\).
2. Translate \((x_{vmin}, y_{vmin})\) to \((x_{umin}, y_{umin})\).

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

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

ii. Clipping window to normalized square (second approach):

1. Apply transformation above twice. First w/ viewport as \((-1, -1), (1, 1))\).
2. Second window as \((-1, -1), (1, 1))\). Why: simplifies computations.

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

\(x_{umin} \leq x \leq x_{umax}\) and \(y_{umin} \leq y \leq y_{umax}\) must be satisfied, else throw away.
D. Line Clipping

1. First Idea: Compute intersections and limit lines using clipping region.

2. Second Idea: Determine which lines are entirely inside or outside. Only compute intersections where necessary. How to apply point test to endpoints, if both endpoints 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.

To determine intersections, use parametric form of line

\[ x = x_0 + t \Delta x \quad \text{and} \quad y = y_0 + t \Delta y \quad 0 \leq t \leq 1 \]

Substitute boundary into one equation and solve for t. If t is outside, the line does not intersect the boundary. Otherwise part of the line is inside. Continue on all edges until the entire line is determined.

However, still slow and poorly formulated.

- Cohen-Sutherland Line Clipping

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

<table>
<thead>
<tr>
<th>Bit Values</th>
<th>Region Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>0000</td>
</tr>
<tr>
<td>0100</td>
<td>0001</td>
</tr>
<tr>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>0001</td>
<td>0011</td>
</tr>
</tbody>
</table>

b. Inside-outside check - if both codes 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 -> inside
 Logical AND results in not(0000) i.e. true -> outside

C. Clipping proceeds to check boundaries in order if bit codes are different on endpoints we compute intersection point, draw away outside portion, repeat the process, and take care of outside. Repeat for all boundaries.

1. Faster algo exist but outside scope of lecture.