Lecture 04: Viewing (Chapter 8, 10)

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 viewable and therefore should be rendered

B. Lesson: more primitives = less 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 viewable.

II. Two-Dimensional Viewing (Chapter 8)

A. Two-Dimensional Viewing terms

i. Clipping window - section of 2D world (scene) selected for display

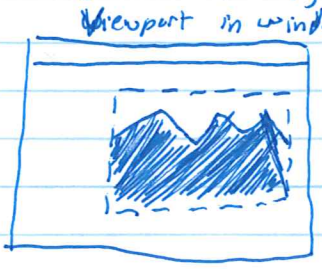
ii. Clipping - Procedurally eliminating portions of a picture that are inside/outside a specified region of space. Usually a rectangle is considered as the region

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

Clipping ⇒ $(xw_{min}, yw_{min}), (xw_{max}, yw_{max})$ Viewing ⇒ $(xv_{min}, yv_{min}), (xv_{max}, yv_{max})$

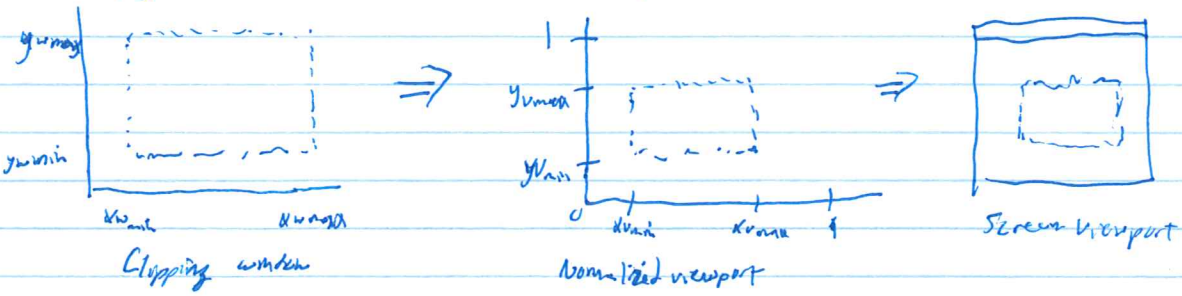


vi. In general - Clipping/viewport could be any shape and multiple viewports are allowed in applications, e.g. CAD.

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 viewport ~~Encode~~ ~~Decode~~



To transform any point (x_w, y_w) from clip window to viewport (x_v, y_v) the following must hold

$$\frac{x_v - x_{v_{min}}}{x_{v_{max}} - x_{v_{min}}} = \frac{x_w - x_{w_{min}}}{x_{w_{max}} - x_{w_{min}}} \quad \text{and} \quad \frac{y_v - y_{v_{min}}}{y_{v_{max}} - y_{v_{min}}} = \frac{y_w - y_{w_{min}}}{y_{w_{max}} - y_{w_{min}}}$$

Solving for (x_v, y_v) we get

$$x_v = s_x x_w + t_x \qquad y_v = s_y y_w + t_y \qquad \text{where}$$

$$s_x = \frac{x_{v_{max}} - x_{v_{min}}}{x_{w_{max}} - x_{w_{min}}} \qquad t_x = \frac{x_{w_{max}} x_{v_{min}} - x_{w_{min}} x_{v_{max}}}{x_{w_{max}} - x_{w_{min}}}$$

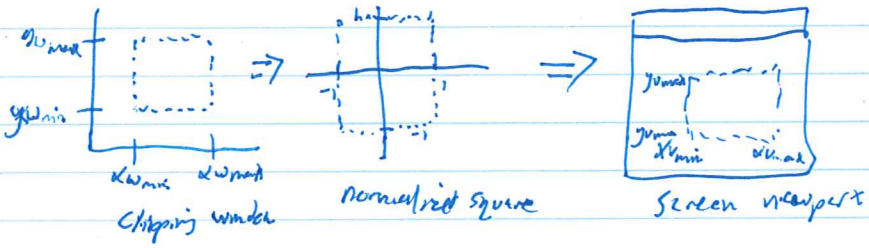
$$s_y = \frac{y_{v_{max}} - y_{v_{min}}}{y_{w_{max}} - y_{w_{min}}} \qquad t_y = \frac{y_{w_{max}} y_{v_{min}} - y_{w_{min}} y_{v_{max}}}{y_{w_{max}} - y_{w_{min}}}$$

So (1) apply scale w/ fixed point $(x_{w_{min}}, y_{w_{min}})$ (2) Translate $(x_{w_{min}}, y_{w_{min}})$ to $(x_{v_{min}}, y_{v_{min}})$

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

i. 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)



essentially apply transformation from above twice. First w/ viewport as $(-1,-1), (+1,+1)$ second w/ window as $(-1,-1), (+1,+1)$. Why: simplifies computations.

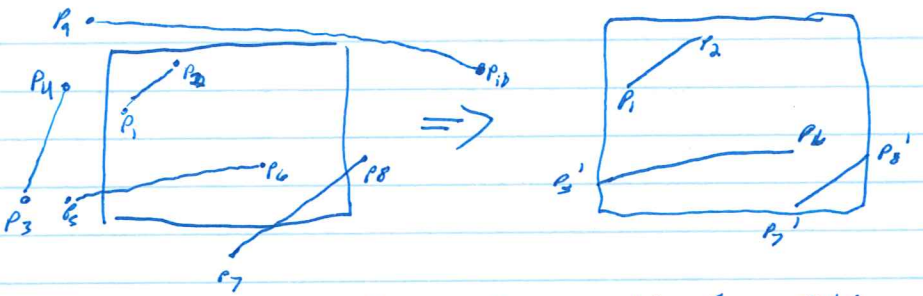
C. Point Clipping *Ask class how to do this*

$x_{w_{min}} \le x \le x_{w_{max}}$ $y_{w_{min}} \le y \le y_{w_{max}}$ must be satisfied, else throw away.

D. Line Clipping.

i. First Idea: Compute intersections of all lines w/ clipping region. *Asks problems? slow bc intersection computation*

ii. Second Idea: Determine which lines are entirely inside vs 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.



To determine intersections use parametric form of line

$$x = x_0 + u\Delta x \qquad y = y_0 + u\Delta y \qquad 0 \le u \le 1$$

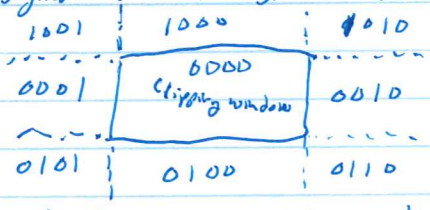
Substitute boundary into one equation and solve for u. If u is outside [0,1] the line does not intersect that border, otherwise part of the line is inside. Continue on all borders until inside/outside is determined

However, still slow and poorly formulated.

iii. Cohen-Sutherland Line Clipping

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

Bit 4	Bit 3	Bit 2	Bit 1
Top	Bottom	Right	Left



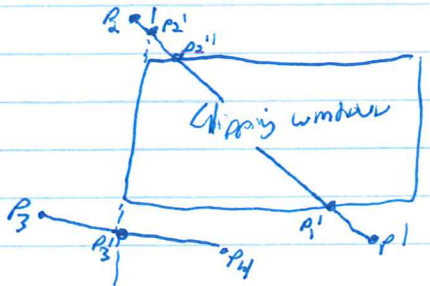
Bit values are assigned using the sign of the difference to the boundary instead of inequality testing. e.g. sign $(x - xw_{min})$

b. Inside-outside check - if both points 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 proceed to check borders left to right if bit codes are different on endpoints we compute intersection point, throw away outside portion, retest the new endpoint and line on inside outside check, Repeat for all boundaries



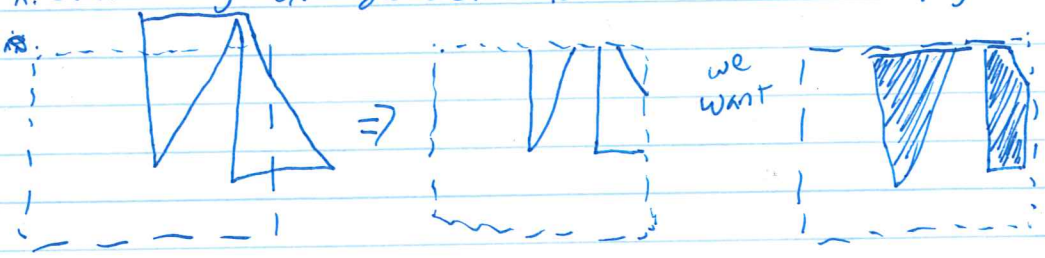
$P_3 - P_4$ – clip to P_3' then both outside

$P_2 - P_1$ clip to P_2' or P_1 clip to P_2'' P_1' clip to P_2'' P_1' then both inside.

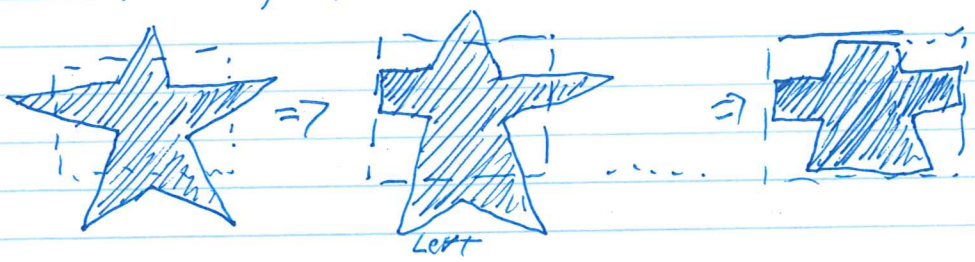
d. Faster algs exist but outside scope of lecture.

E. Polygon-fill Clipping

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



ii. Can progressively clip polygon ~~edges~~ 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.

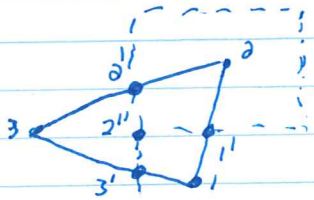
iv. Sutherland-Hodgeman Polygon Clipping

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. ~~Squeeze~~

b. Rules to pass ~~vertices~~ 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 send only the intersection point
4. if both are outside, send ~~not~~ vertices on

c. Example



d. Discussion * ask * ~~Downsides~~ - parallelizable only one output poly extraneous lines

Input edge →	left →	right →	Bottom →	Top
(1,2) :	in/in → {2}	{2,2'} : in/in → {2'}	{1,3'} : in/out → {2''}	{2'',1'} : in/in → {1'}
{2,3} :	in/out → {2'}	{2',3''} : in/in → {3''}	{3'',1'} : out/out → { }	{1',2} : in/in → {2}
{3,1} :	out/in → {3',1}	{3'',1'} : in/in → {1'}	{1',2} : out/in → {1',2}	{2,2'} : in/in → {2'}
		{1',2} : in/in → {2}	{2',2''} : in/in → {2''}	{2',2''} : in/in → {2''}

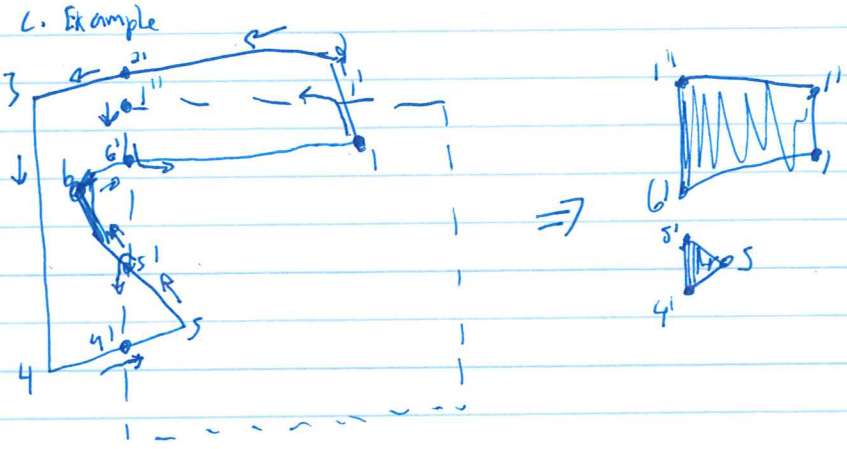
V. Weiler-Atherton Polygon Clipping

a. Basic idea - trace polygon boundary to produce clipped fill region

b. In a counterclockwise order

1. Process until in/out 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

c. Example



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

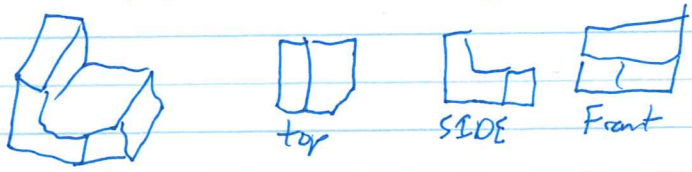
III. 3D Viewing. (chapter 10)

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 drafting/CAD, parallel lines stay parallel



b. Perspective projection - lines converge in distance. Mimics reality

iii. Depth Cueing - Need to know what is front/back of an object. Closer lines appear brighter or more intense

iv. Visible object detection - depth relationships of an object being in front (Ch. 16)

v. Surface rendering - lighting/detailing surfaces (Ch. 17)

vi. more such as stereoscopic viewing for 3D glasses

B. Three-dimensional viewing pipeline

MC → Transforms → WC → Viewing Transform → View coordinates →

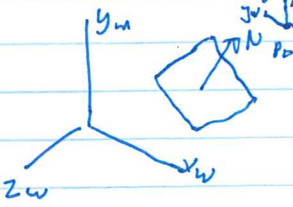
→ Projection Transformation → Projection coordinates → Normalization Transformation and Clipping → normalized coordinates → Viewport Transform → Device coordinates

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 by picking a position "at" to look at. camera at. "look-at-point"



Zp defines how far on the z-axis the plane is.

iii. Pick an up vector $\vec{V}$ to define y-axis of view coordinates (not parallel to $\vec{N}$)

iv. uvn reference frame (get full axis information)

$$\hat{n} = \frac{\vec{N}}{|\vec{N}|} = (n_x, n_y, n_z) \quad \hat{u} = \frac{\vec{V} \times \vec{n}}{|\vec{V} \times \vec{n}|} = (u_x, u_y, u_z) \quad \vec{v} = \vec{n} \times \vec{u} = (v_x, v_y, v_z)$$

z-axis x-axis y-axis

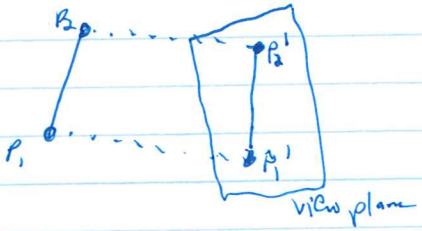
D. Transforming from world coordinates to viewing coordinates

i. Translate view origin to world origin

ii. Rotate to align coordinate frame to world frame

$$T = \begin{bmatrix} 1 & 0 & 0 & -x_0 \\ 0 & 1 & 0 & -y_0 \\ 0 & 0 & 1 & -z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad R = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ n_x & n_y & n_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M = RT = \begin{bmatrix} u_x & u_y & u_z & -u \cdot P_0 \\ v_x & v_y & v_z & -v \cdot P_0 \\ n_x & n_y & n_z & -n \cdot P_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

E. Orthogonal Projections - transform of objects along parallel lines to view plane



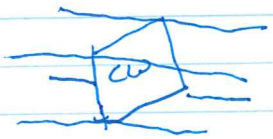
(Recall we have oriented our camera)

i. Axonometric - display more than one face
Isometric - axonometric such that view plane intersects each coordinate axis at the same distance from the origin

ii. Orthogonal Projection Coordinates
a. Direction of projection is parallel to z_{view} axis, so it is trivial
b. projection

$x_p = x \quad y_p = y \quad z$ is preserved for visibility

iii. Orthogonal-projection view volume is
a. sides of clipping window define infinite region of clipping



b. We specify near-far clipping planes to limit extent. Since direction is negative z_{view} axis $z_{far} < z_{near}$
c. Shape is rectangular parallelepiped

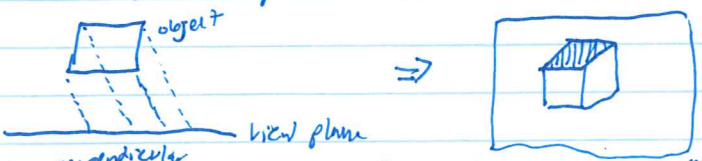
iv. Normalization transformation - transform to symmetric cube $(-1,1)$ for clipping

$$M_{ortho,norm} = \begin{bmatrix} \frac{2}{x_{w\,max} - x_{w\,min}} & 0 & 0 & \frac{-x_{w\,max} + x_{w\,min}}{x_{w\,max} - x_{w\,min}} \\ 0 & \frac{2}{y_{w\,max} - y_{w\,min}} & 0 & \frac{-y_{w\,max} + y_{w\,min}}{y_{w\,max} - y_{w\,min}} \\ 0 & 0 & \frac{-2}{z_{near} - z_{far}} & \frac{z_{near} + z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

z is negative to flip to left handed coordinates. Otherwise: z_{near} needs to be -1

multiply on right of model-view transformation, then can apply to object all at once.

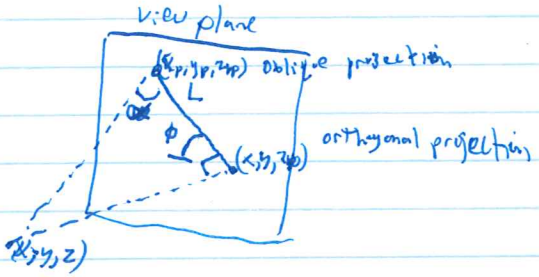
F. Oblique Parallel Projection - parallel lines in a direction relative to view plane



when it isn't ~~perpendicular~~ to view plane it is called an oblique parallel projection

For combined Top, front, side view

i. Specification - need two angles α, ϕ



L - length of view plane line $(x_p, y_p, z_{vp}) \to (x, y, z_{vp})$

α - intersection angle

or

$(x, y, z) \to (x_p, y_p, z_{vp})$ with $(x_p, y_p, z_{vp}) \to (x, y, z_{vp})$

ϕ - angle of view plane line from horizontal

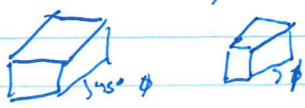
So $x_p = x + L\cos\phi$ $\quad y_p = y + L\sin\phi$ } represents Shearing projection

$\tan\alpha = \frac{z_{vp} - z}{L}$ $\quad L = \cot\alpha\ (z_{vp} - z)$

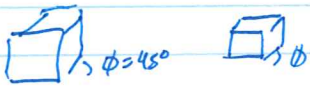
an orthogonal projection has $\alpha = 90°$

ii. Types - ϕ is usually 30° or 45° $\tan\alpha$ is either 1 or 2.

$\alpha = 45°$ is a cavalier projection.

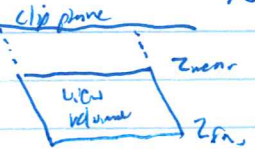
cavalier 

$\tan\alpha = 2$ is a cabinet projection

cabinet 

iii. Easier specification - parallel projection vector $\vec{V_p}$

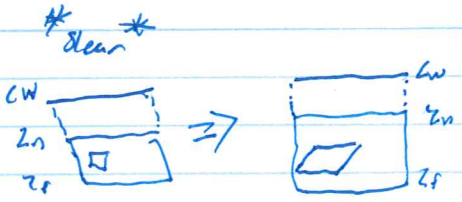
we can derive $x_p = x + (z_{vp} - z)\frac{V_{px}}{V_{pz}}$ $\quad y_p = y + (z_{vp} - z)\frac{V_{py}}{V_{pz}}$

iv. View volume 

Shape: oblique parallelepiped

v. Transform

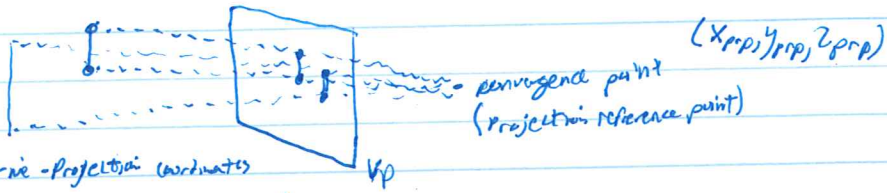
$$M_{oblique} = \begin{bmatrix} 1 & 0 & -\frac{V_{px}}{V_{pz}} & z_{vp}\frac{V_{px}}{V_{pz}} \\ 0 & 1 & -\frac{V_{py}}{V_{pz}} & z_{vp}\frac{V_{py}}{V_{pz}} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



vi. Same mapping to normalized cube

$M_{obliquestrans} = M_{orthopersp} M_{oblique}$

G. Perspective Projection – parallel lines converge, more realistic
distant objects appear smaller



i. Perspective-Projection coordinates

z_{vp} is z of view plane

So from parametric form of line $0 \le u \le 1$

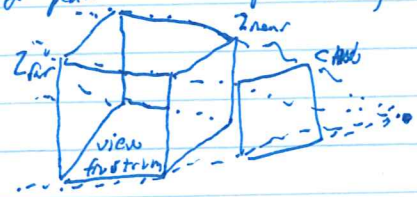
$$u = \frac{z_{vp} - z}{z_{prp} - z} \quad \text{and} \quad x_p = x\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + x_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right)$$

$$y_p = y\left(\frac{z_{prp} - z_{vp}}{z_{prp} - z}\right) + y_{prp}\left(\frac{z_{vp} - z}{z_{prp} - z}\right)$$

Problem: denominators are function 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 view plane image is inverted *

ii. View volume

a. Boundary planes are not parallel anymore, so we have an infinite rectangular pyramid



w/ near-far planes we have a frustrum

iii. Perspective-Projection Transformation Matrix

a. Because of z coordinate dependency, we can exploit homogeneous coordinates

$$x_p = \frac{x_h}{h} \quad y_p = \frac{y_h}{h} \quad \text{and } h \text{ is } z_{prp} - z$$

so $x_h = x(z_{prp} - z_{vp}) + x_{prp}(z_{vp} - z)$ $\quad y_h = y(z_{prp} - z_{vp}) + y_{prp}(z_{vp} - z)$

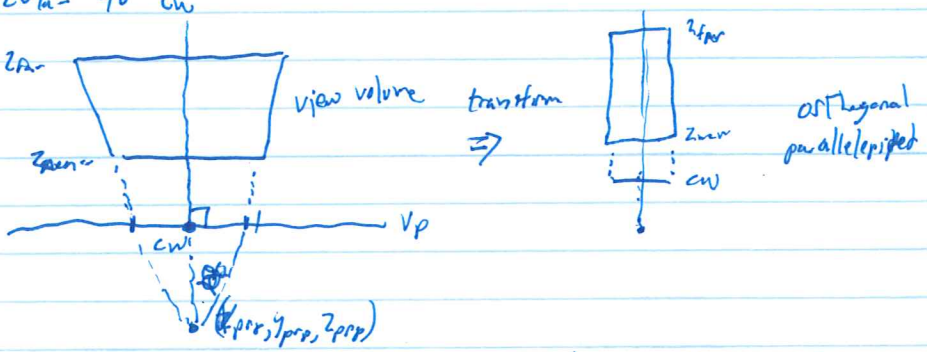
b. So $p_h = M_{persp} \cdot P$ many choices for M_{persp} but we need one to preserve z information.

$$M_{persp} = \begin{bmatrix} z_{prp} - z_{vp} & 0 & -x_{prp} & x_{prp} z_{prp} \\ 0 & z_{prp} - z_{vp} & -y_{prp} & y_{prp} z_{prp} \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & z_{prp} \end{bmatrix}$$

s_z, t_z dependent on normalization range.

c. Converts scene to homogeneous parallel projection coordinates, i.e. an oblique projection, so we have 2 cases symmetric frustrum, i.e. orthogonal, or not, i.e. oblique.

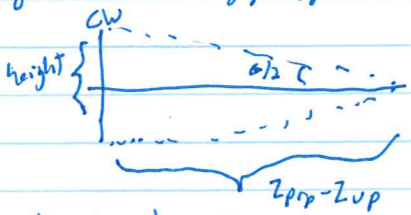
iv. Symmetric Perspective-projection frustrum - line through center of frustrum is perpendicular to cw



a. in this case, cw coordinates can be expressed in window coordinates

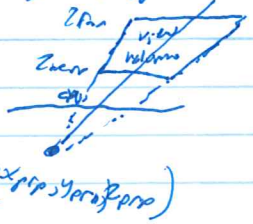
b. We can specify by field-of-view angle + aspect ratio

c. cw height is: $2(z_{prp} - z_{vp}) \tan\left(\frac{\theta}{2}\right)$ and diagonal elements of matrix are updated to $z_{prp} - z_{vp} = \text{height}/2 \cot(\theta/2) = \frac{\text{width} \cdot \cot(\theta/2)}{2 \cdot \text{aspect}}$



d. After apply normalization transform

v. Oblique perspective-Projection Frustrum



Need Shear. Simplify through fixed reference at origin.

$$M_{obliquepers} = M_{pers} M_{zshear} = \begin{bmatrix} -z_{near} & 0 & \frac{xw_{min} + xw_{max}}{2} & 0 \\ 0 & -z_{near} & \frac{yw_{min} + yw_{max}}{2} & 0 \\ 0 & 0 & s_z & t_z \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

vi. After perspective projection, divide coordinates by homogeneous parameter h

Frustrum transformed to orthogonal parallelpiped

a. So we need to scale x,y coordinates + determine s_z, t_z

b. Actual perspective matrix becomes

$$M_{normpersp} = \begin{bmatrix} \frac{-2z_{near}}{xw_{max} - xw_{min}} & 0 & \frac{xw_{max} + xw_{min}}{xw_{max} - xw_{min}} & 0 \\ 0 & \frac{-2z_{near}}{yw_{max} - yw_{min}} & \frac{yw_{max} + yw_{min}}{yw_{max} - yw_{min}} & 0 \\ 0 & 0 & \frac{z_{near} + z_{far}}{z_{near} - z_{far}} & \frac{-2z_{near} z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

and

$$M_{normsymmpers} = \begin{bmatrix} \frac{\cot(\theta/2)}{aspect} & 0 & 0 & 0 \\ 0 & \cot(\frac{\theta}{2}) & 0 & 0 \\ 0 & 0 & \frac{z_{near} + z_{far}}{z_{near} - z_{far}} & \frac{-2z_{near} z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

multiply this on left of view transform then apply clipping.

H. Viewport to device transform

$$M_{\text{normview to device}} = \begin{bmatrix} \frac{x_{V_{max}} - x_{V_{min}}}{2} & 0 & 0 & \frac{x_{V_{max}} + x_{V_{min}}}{2} \\ 0 & \frac{y_{V_{max}} - y_{V_{min}}}{2} & 0 & \frac{y_{V_{min}} + y_{V_{max}}}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

z coordinate is mapped to depth buffer