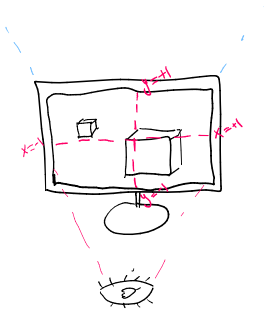
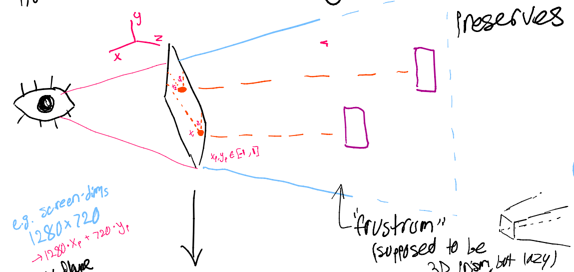
Projection-Based Rendering

Tuesday, August 6, 2024 11:34 AM



How to write a mapping equation (from $\begin{bmatrix}x\\y\\z\end{bmatrix} \to \begin{bmatrix}x_p\\y_p\end{bmatrix}$) that preserves information about depth?



(Aside for determining frustum plane intercept points) *see update to frustum culling*

Let $y_p = \frac{\text{height}}{2} \cdot \frac{y_h}{\tan(\frac{\alpha}{2}) \cdot z_n}$

$\Rightarrow \frac{2}{\text{height}} \cdot y_p = \tan(\frac{\alpha}{2}) \cdot z_n \Rightarrow y_h = \frac{\text{height} \cdot \tan(\frac{\alpha}{2}) \cdot z_n}{2}$

$x_p = \frac{\text{width}}{2} = \frac{x_h}{\tan(\frac{\alpha}{2}) \cdot z_n}$

$\Rightarrow \frac{2}{\text{width}} \cdot x_h = \tan(\frac{\alpha}{2}) \cdot z_n \Rightarrow x_h = \frac{\text{width} \cdot \tan(\frac{\alpha}{2}) \cdot z_n}{2}$

Top/left ($\nearrow$)

$0 = \frac{\text{width} \cdot X}{\tan(\frac{\alpha}{2}) \cdot z_n} + \frac{\text{width}}{2}$

$-\frac{\text{width} \cdot \tan(\frac{\alpha}{2}) \cdot z_n}{2} = \text{width} \cdot X$

$\Rightarrow X = -\frac{\tan(\frac{\alpha}{2}) \cdot z_n}{2} = -\frac{1}{2}$

Bottom/right

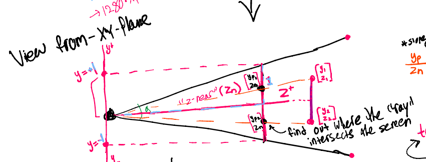
$\text{width} = \frac{\text{width} \cdot X}{\tan(\frac{\alpha}{2}) \cdot z_n} + \frac{\text{width}}{2}$

$\Rightarrow \frac{1}{2} = \frac{X}{\tan(\frac{\alpha}{2}) \cdot z_n}$

$\Rightarrow X = \frac{\tan(\frac{\alpha}{2}) \cdot z_n}{2} = \frac{1}{2}$

e.g. screen-dims 1280x720

$\rightarrow 1280 \cdot x_p + 720 \cdot y_p$



same same (as the other one)

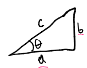
$\frac{y_p}{z_n} = \frac{y}{z} \Rightarrow y_p = \frac{y \cdot z_n}{z}$

$\tan(\frac{\alpha}{2}) = \frac{1}{z_n} \Rightarrow z_n = \frac{1}{\tan(\frac{\alpha}{2})}$

$\therefore y_p = \frac{y \cdot (\frac{1}{\tan(\frac{\alpha}{2})})}{z} = \frac{y}{\tan(\frac{\alpha}{2}) \cdot z}$

by an entirely symmetric argument:

$x_p = \frac{x \cdot (\frac{1}{\tan(\frac{\alpha}{2})})}{z} = \frac{x}{\tan(\frac{\alpha}{2}) \cdot z}$



$\sin\theta = \frac{\text{opp}}{\text{hyp}} = \frac{b}{c}$

$\cos\theta = \frac{\text{adj}}{\text{hyp}} = \frac{a}{c}$

$\tan\theta = \frac{\text{opp}}{\text{adj}} = \frac{b}{a}$

$\cdot \tan(\frac{\alpha}{2}) = \frac{1}{z_n}$

Now just define some points in $\mathbb{R}^{3\times1}$ and define their projection onto screen as:

Let $w :=$ display width

$h :=$ display height

$\alpha :=$ Field of view (FOV)

$x, y \in [-1, 1]$ (constrained to screen dimensions), $z \in \mathbb{Z}$

Then display coordinates are:

$$\begin{bmatrix} w \cdot \frac{x}{\tan(\frac{\alpha}{2}) \cdot z} \\ h \cdot \frac{y}{\tan(\frac{\alpha}{2}) \cdot z} \end{bmatrix}$$

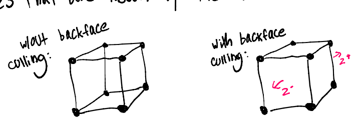
- how are we going to specify drawing at specific pixels? – just use ratio / screen constants
- how to connect $\mathbb{R}^{3\times1}$ points to draw shapes?
  - ↳ could define functions to draw point inbetween 2-points, but maybe too many points? How to determine LOD?

Backface culling

- why? while not initially noticable when rendering a 8-point cube, rendering 3D models from internet (w/thousands of vertices) tanks performance from ~55 fps to a whopping ~9 fps.

When we are specifying the vertices of 3D geometry in 3D space to render, how shall we group vertices. My first thought was simply to pair any two points that I want to draw a line between — why overcomplicate it with all the triangle business that I keep hearing about? Well, it turns out there's a good reason and it became evident in implementation of backface culling.

The idea of backface culling is to not render any geometric faces that should be hidden from view (ie. faces that are hidden by the front faces); for example:



However, how should we define faces? And what deems any given face a backface and not a camera facing front-face?

Lets let a face be a specified group of points that lie on the same plane.

As you might remember from multi-variable calc. it take 3 points to uniquely define a plane;

Lets let a face be a specified group of points that lie on the same plane. As you might remember from multi-variable calc. it take 3 points to uniquely define a plane; this is why triangles are the bases of 3D models. Therefore,

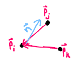
given a set $A := \{[\begin{smallmatrix}x_1\\y_1\\z_1\end{smallmatrix}], [\begin{smallmatrix}x_2\\y_2\\z_2\end{smallmatrix}], \dots [\begin{smallmatrix}x_n\\y_n\\z_n\end{smallmatrix}]\}$, where $n \geq 3$ (m∈{1,...,n})

and $\vec{n} :=$ normal vector defining shared plane by $\forall [\begin{smallmatrix}x_m\\y_m\\z_m\end{smallmatrix}] \in A$,

then we can uniquely determine $\vec{n}$ in the following way:

let $i,j,k \in \{1,\dots,n\}$ s.t. $i \neq j \neq k$ (let $\vec{P}_i = [\begin{smallmatrix}x_i\\y_i\\z_i\end{smallmatrix}]$)

& $\vec{n} = [\begin{smallmatrix}x^*\\y^*\\z^*\end{smallmatrix}]$



*perpendicular (to both)

$$\left\langle \begin{bmatrix}x_j-x_i\\y_j-y_i\\z_j-z_i\end{bmatrix}, \begin{bmatrix}x_k-x_i\\y_k-y_i\\z_k-z_i\end{bmatrix} \right\rangle \quad \left.\begin{matrix}\vec{n}\cdot(\vec{P}_j-\vec{P}_i)=0 \;\&\\ \vec{n}\cdot(\vec{P}_k-\vec{P}_i)=0\end{matrix}\right\} \Rightarrow \begin{matrix}x^*(x_j-x_i)+y^*(y_j-y_i)+z^*(z_j-z_i)=0\\ x^*(x_k-x_i)+y^*(y_k-y_i)+z^*(z_k-z_i)=0\end{matrix}$$

$$\Rightarrow \left.\begin{matrix}z_2(x^*x_1+y^*y_1+z^*z_1)=0\\ -z_1(x^*x_2+y^*y_2+z^*z_2)=0\end{matrix}\right\} = \begin{matrix}x^*x_1z_2+y^*y_1z_2+z^*z_1z_2=0\\ -\;x^*x_2z_1+y^*y_2z_1+z^*z_2z_1=0\\ \hline x^*(x_1z_2-x_2z_1)+y^*(y_1z_2-y_2z_1)=0\end{matrix}$$

$$\begin{vmatrix} i & j & k\\ x_1 & y_1 & z_1\\ x_2 & y_2 & z_2\end{vmatrix} = i(y_1z_2-y_2z_1)-j(x_1z_2-x_2z_1)+k(x_1y_2-x_2y_1)$$

$\therefore x^* = x_1z_2-x_2z_1,\; y^* = y_1z_2-y_2z_1,$

$z^* = x_1y_2-x_2y_1$, *by a symmetric argument (or plug x^*/y^* into eq. equations)

we can now determine the normal vector that defines the plane the set of points rest on.

Now, if the z component of the normal is positive (per our [arbitrary] convention) it is pointing out away from the camera & therefore not to be rendered. If it's negative, face/plane is facing view & we should draw it:

```
for every plane (i.e. every triange in mesh):
    if( z* = x1y2-x2y1 = (xj-xi)(yk-yi) - (xk-xi)(yj-yi) < 0):
        drawEdgesConnectingVerticesInPlane(A)
    else:
        continue #else statenement just to be verbose
```

insert (clip only if not backfaced)

Frustrum Culling:

Define 6 planes to bound the viewing frustrum:

- Near plane

a point that makes this true belongs to the plane

where we want near plane relative to view window

$$\langle 0,0,1\rangle \cdot (\langle x,y,z\rangle - \langle 0,0,z^*_{near}\rangle) = 0$$

- Far plane (optional—like a render dist.)

direction doesn't matter, same axis

$$\langle 0,0,1\rangle \cdot (\langle x,y,z\rangle - \langle 0,0,z^*_{far}\rangle) = 0$$

- Top plane

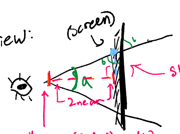
$\langle 0, 2n, \tfrac{1}{2}\rangle$

~~$\langle 0, -\sin(\tfrac{a}{2}-90), \cos(\tfrac{a}{2}-90)\rangle$~~ $\cdot (\langle x,y,z\rangle - \langle 0, \tfrac{1}{2}, z_{near}\rangle)$

~~(possibly just the height of screen?)~~ *by frustum intersect calculation above*

x-axis rotation

y-z view:



slope $= \frac{\frac{1}{2}\text{height}}{z_{near}}$

$180° - (\tfrac{a}{2})° - 90° = b°$

how to find vector that is exactly b° rotated about point $[\begin{smallmatrix}\frac{1}{2}height\\ z_{near}\end{smallmatrix}]$ along the y-z axis?

slope $= \frac{\text{height}}{z_{near}}$

- Update on plane normal vectors:
  - previous approach occluded too late
  - new (simpler) approach: simply rotate vectors $\langle 0, -\tfrac{1}{2}, z_{near}\rangle$ (of frustum) (y-z: rise/run) / $\langle \tfrac{1}{2}, 0, z_{near}\rangle$ (x-z) by $-90°$ to achieve normal to frustum $\Rightarrow \langle 0, z_{near}, \tfrac{1}{2}\rangle$ (y-z) / $\langle z_{near}, 0, \tfrac{1}{2}\rangle$ (x-z)

Turn $(\tfrac{a}{2})°$ & then $-90°$ for right angle form this

x-axis rotation matrix

$$\begin{bmatrix}1&0&0\\0&\cos(\theta)&-\sin(\theta)\\0&\sin(\theta)&\cos(\theta)\end{bmatrix}\begin{bmatrix}0\\0\\1\end{bmatrix} = \begin{bmatrix}0\\-\sin(\theta)\\\cos(\theta)\end{bmatrix} \therefore \theta = (\tfrac{a}{2})° - 90°$$

- Bottom Plane: $\langle 0, -2n, \tfrac{1}{2}\rangle$ ~~$\langle 0, -\sin(90-\tfrac{a}{2}), \cos(\tfrac{a}{2}-90)\rangle$~~ $\cdot (\langle x,y,z\rangle - \langle 0, -\tfrac{1}{2}, z_{near}\rangle)$

y-axis rotation matrix

- Right plane $(\theta = \tfrac{a}{2} - 90)$

• Bottom Plane: $\langle 0, -\sin(90-\frac{a}{2}), \cos(\frac{a}{2}-90)\rangle \cdot (\langle x,y,z\rangle - \langle 0, \frac{-1/2}{2}, 2near\rangle)$

y-axis rotation matrix

$$\begin{bmatrix} \cos(\theta) & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \sin\theta \\ 0 \\ \cos\theta \end{bmatrix}$$

• Right Plane ($\theta = \frac{a}{2} - 90$)

y-axis rotation

$\langle \sin(\frac{a}{2}-90), 0, \cos(90-\frac{a}{2})\rangle \cdot (\langle x,y,z\rangle - \langle \frac{X_{width}}{2}, 0, 2near\rangle)$ ($\frac{X_{width}}{2}$ → 1/2)

$\langle 2n, 0, 1/2\rangle$

• Left plane ($\theta = 90 - \frac{a}{2}$)

$\langle -2n, 0, 1/2\rangle$

$\langle -\sin(90-\frac{a}{2}), 0, \cos(\frac{a}{2}-90)\rangle (\langle x,y,z\rangle - \langle -\frac{X_{width}}{2}, 0, 2near\rangle)$ ($\frac{X_{width}}{2}$ → -1/2)

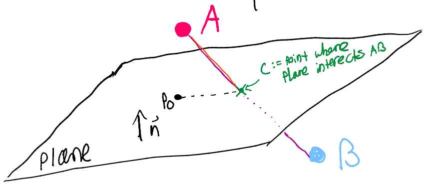
Now...

```
for (polygon in mesh):          # i.e. triangle (probably) points
    newPoly = sutherlandClipPolygon(polygon, {Near, Far, Top, Bottom, Right, Left})   # (planes to clip against)
    polygon = newPoly
    # for many the update will do nothing, but clipped poly returned
      when necessary
```

To clip, we need to know how to determine whether a given point is infront or behind a plane.

If we draw a line from $A \to P_0 \in$ plane

Let $P_0 \in$ Plane



$\vec{n} = \langle a, b, c\rangle$

$(\vec{C} - \vec{P_0}) \cdot \vec{n} = 0 \Rightarrow \vec{n}\cdot\vec{C} - \vec{n}\cdot\vec{P_0} = 0$

$\vec{A} + (\vec{B} - \vec{A}) \cdot t = C$

$\vec{AB}$ is some scale factor, t, away from C.

$\vec{n} \cdot (\vec{A} + \vec{B}t - \vec{A}\cdot t) - \underbrace{\vec{n}\vec{P_0}}_{D} = 0$

$\Rightarrow \vec{n}\cdot(\vec{A} + \vec{B}t - \vec{A}\cdot t) + \vec{n}\cdot\vec{A} = \vec{n}\cdot\vec{P_0}$?

$\Rightarrow \vec{n}\cdot(\vec{B} - \vec{A})\cdot t + \vec{n}\cdot\vec{A} = \vec{n}\cdot\vec{P_0}$

$\Rightarrow t = \frac{\vec{n}\cdot\vec{P_0} - \vec{n}\cdot\vec{A}}{\vec{n}\cdot\vec{B} - \vec{n}\cdot\vec{A}}$

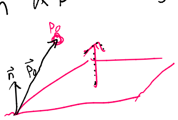
if $t < 1$: P is beyond (infront) plane

if $t > 1$: P is before (behind) plane

if $t == 0$: P is on plane

For vectors $\vec{\alpha}, \vec{\beta}$, if the angle between them is $< 90°$, then $\vec{\alpha}\cdot\vec{\beta} > 0$ (pos.) & negative otherwise

← Ultimately used the dot product of the point with the plane's normal to determine whether a point is infront/behind plane



∴ if $\vec{P_0}\cdot\vec{n} < 0$:
  P_0 is in behind plane
else if $\vec{P_0}\cdot\vec{n} > 0$:
  P_0 is infront of plane
else: // ... == 0
  P_0 is on plane

Other Optimizations

• If screen space already has line through it, don't redraw it