BRAIN

STORM

USAR CLASES.

.obj.

↳ PARSER → vértices
texturas
caras.

RENDER.

triangular con for.

↳  a cada uno aplicar MVP

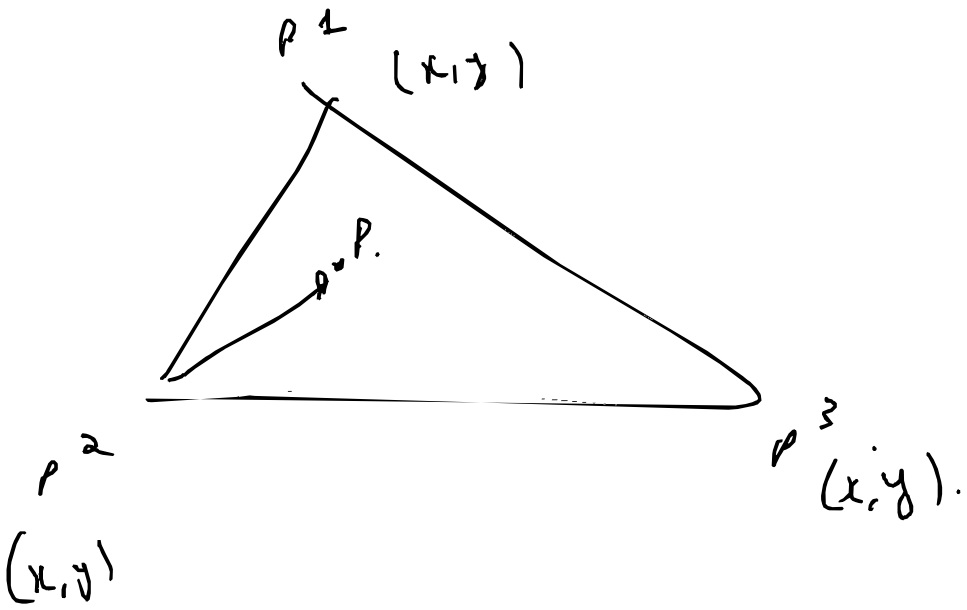
↳ Frustum Culling
Backface culling → Iluminación.
Poner a coordenadas reales

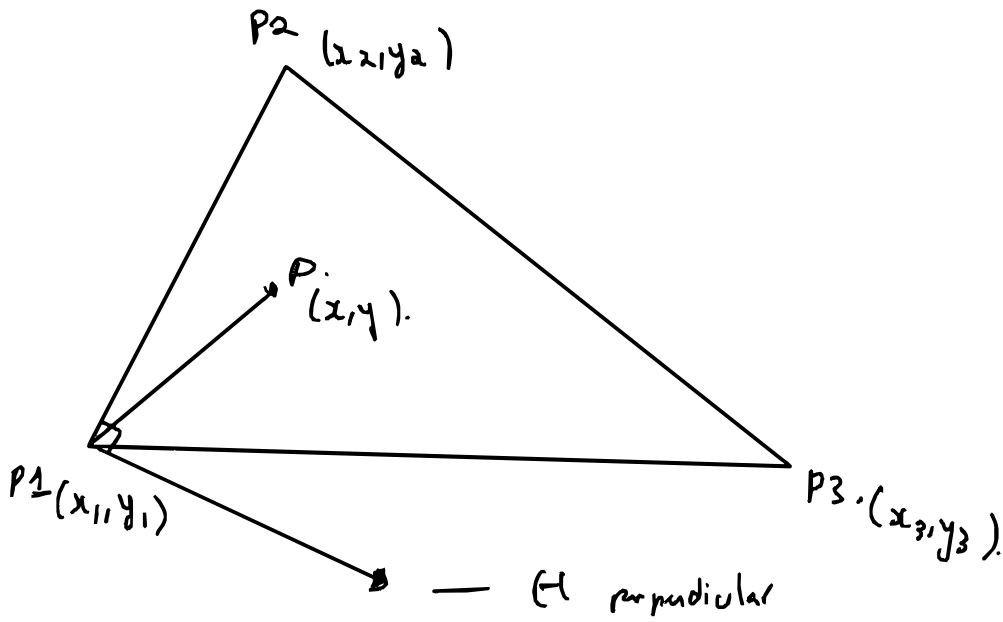
↳ Cada triangulo → MULTITHREADING

↳ Dibujar con Z-Buffer e interpolación

* texturas
* extraer imagenes.

MODEL VIEW PROJECTION





— ⊥ perpendicular a p2 y p1. = L.

Si dot product de P y L de todos es 0 ó neg.
pintar triangulo.

3 CALCULOS.

A y B.

$Y = B - A$. y $Y' = (x,y) \to (y, -x)$

dot product P y Y' y listo :) ⇒ edge 1.

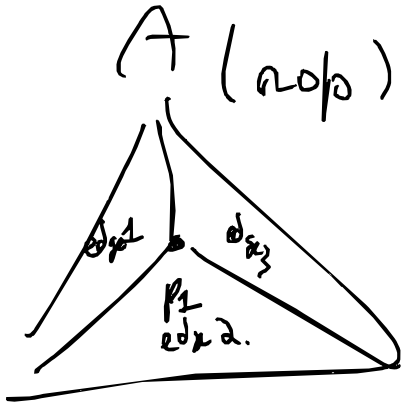
lado 12
x_p
y_p
dot product ⇒

P.
x
y.

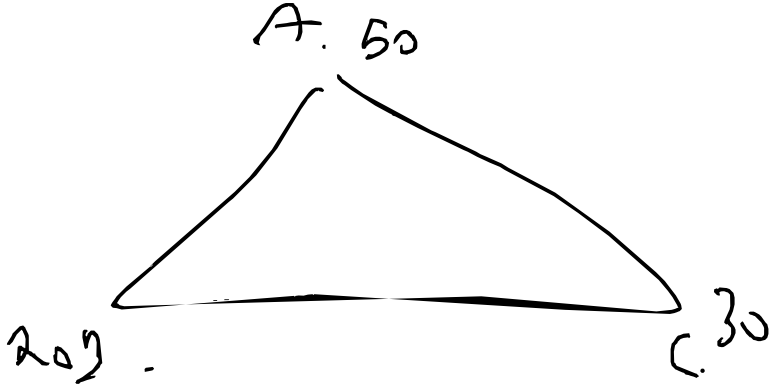
$x x_p + y y_p$.
↓ ↓
y_p $-x_p$.

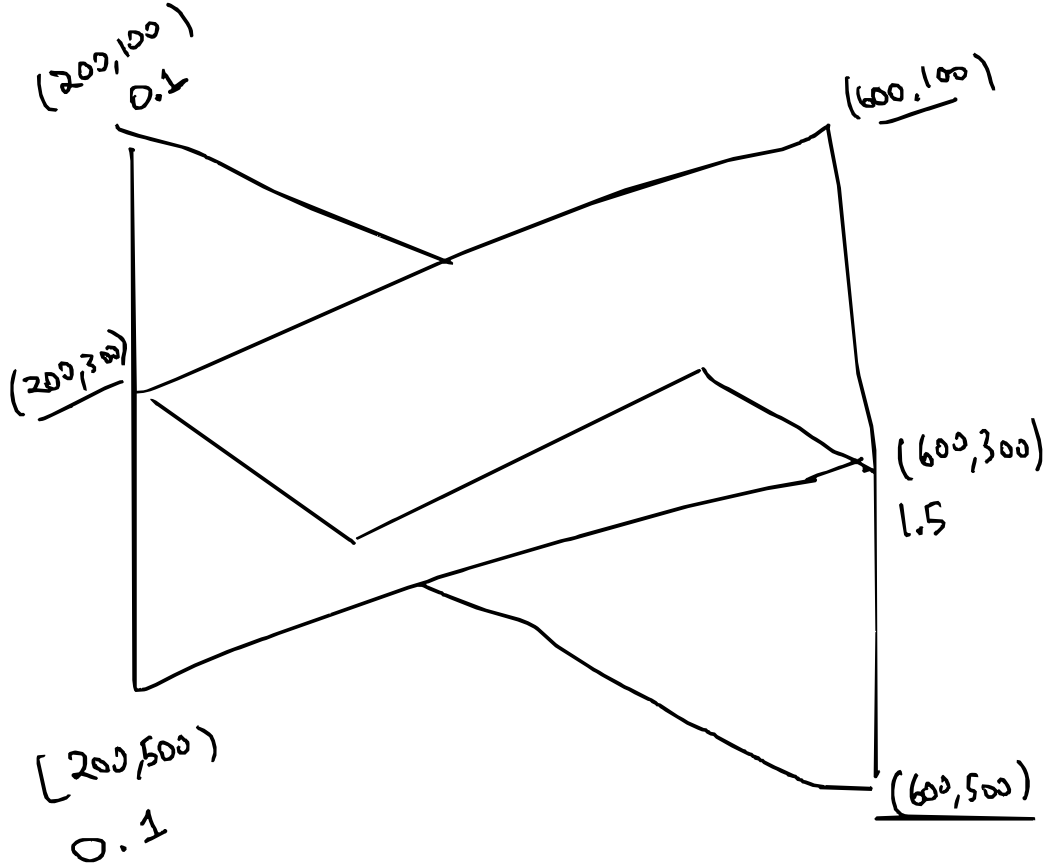
⇒ $x \cdot y_p + y \cdot (-x_p)$.

Color [255, 255, 255]



Norm 1 x red + Norm 2 x azul + Norm 3 x verde.




(200,100)
0.1
(600,100)
(200,300)
(600,300)
1.5
[200,500)
0.1
(600,500)

MVP.

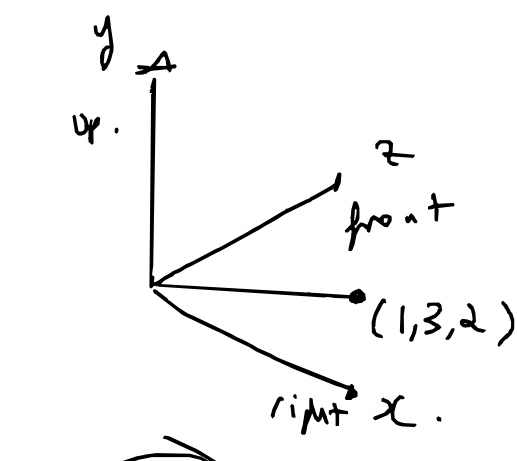
rot x rot y rot z.

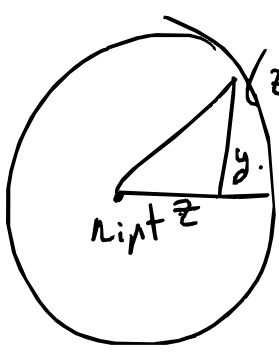
escalar displazar

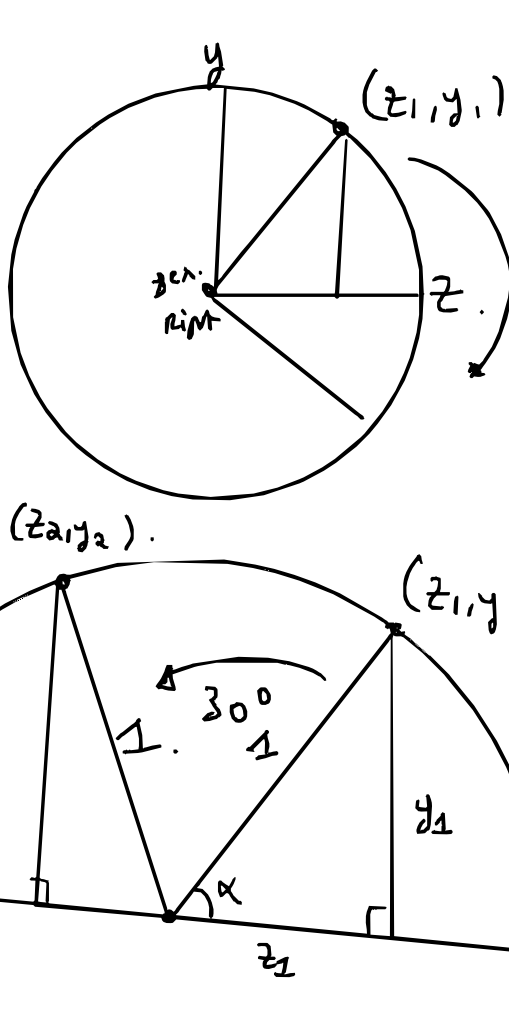
proyectar.

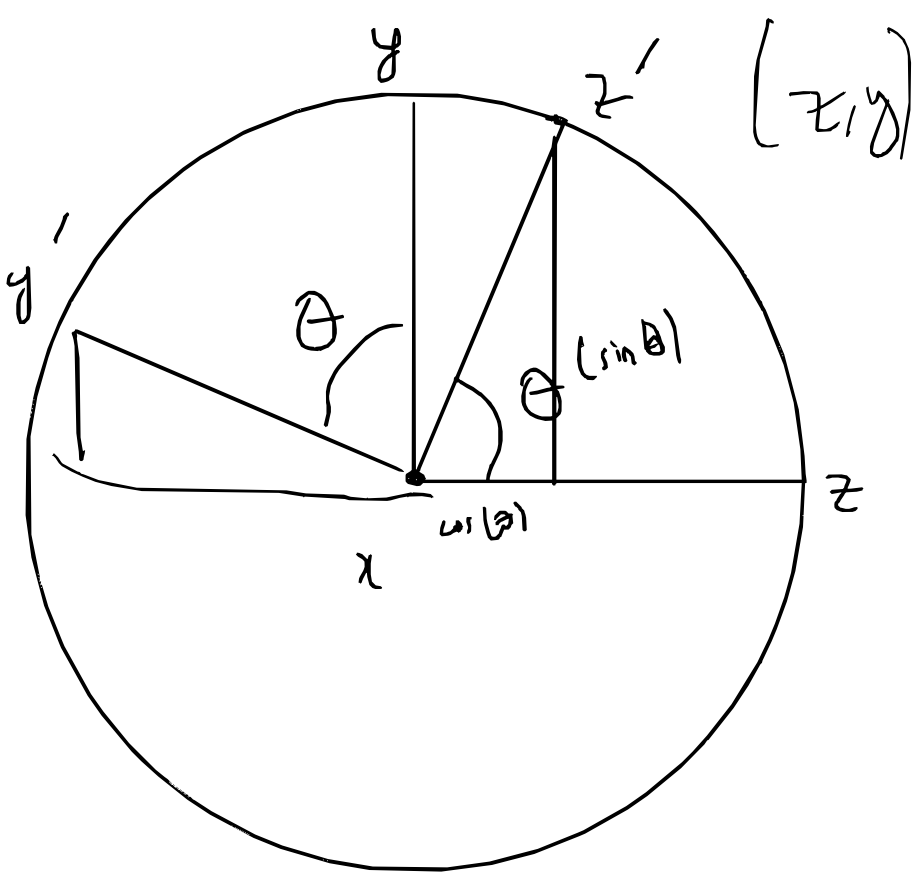
$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} b_{11} \\ b_{12} \\ b_{13} \\ b_{14} \end{bmatrix}$$

$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$ y queremos rotarlo en el eje y. (k grados)









$z \rightarrow (\cos\theta, \sin\theta)$.

$y \rightarrow (-\sin\theta, \cos\theta$

z leaves $\rightarrow (x,$

$x = x$.

$y = z \cdot \sin\theta + y \cos\theta$

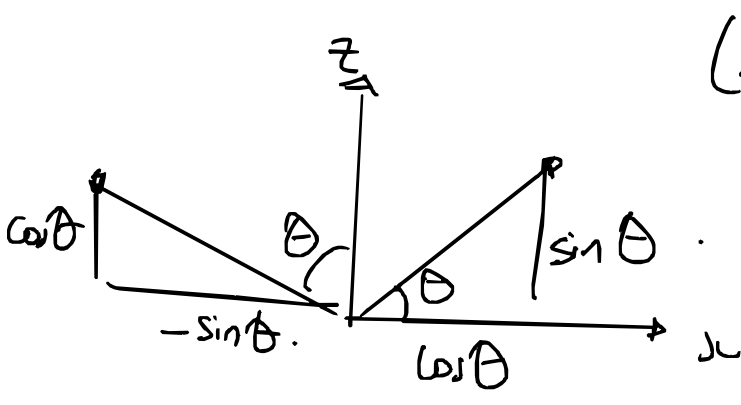
$z = z \cdot \cos\theta + \sin\theta\, y$.

$\begin{pmatrix} x \\ y \\ z \end{pmatrix} \rightarrow x$

$\rightarrow \quad y \cos\theta - z\sin\theta$

$\rightarrow \quad y \cdot \sin\theta + z \cdot \cos\theta$.

ROT x.

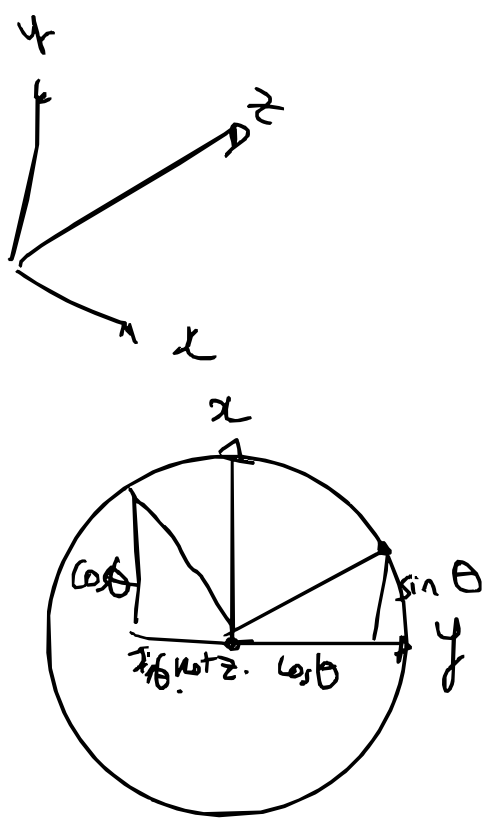


Vec x $\rightarrow$ $\cos\theta x + \sin\theta z$.

Vec y $\rightarrow$ y.

Vec z $\rightarrow$ $-\sin\theta x + \cos\theta z$.

ROT y.



Vec $x = -\sin\theta y + \cos\theta x$

Vec $y = \cos\theta y + \sin\theta x$

Vec $z = z$

ROT z.

$\begin{bmatrix} & & & dx \\ & 1 & & dy \\ & & 1 & dz \\ & & & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$

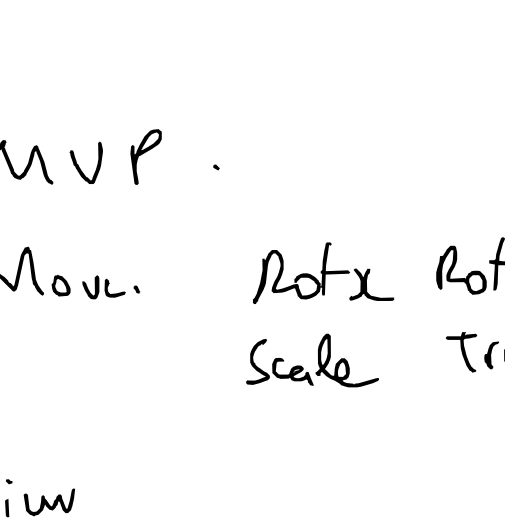
MVP.

Move. Rotx Roty Rotz
Scale translate.

View

Projection.

Lookat matrix.



cam pos, front, up, right.

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \qquad \begin{bmatrix} 1 & 0 & 0 & -pos_x \\ 0 & 1 & 0 & -pos_y \\ 0 & 0 & 1 & -pos_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

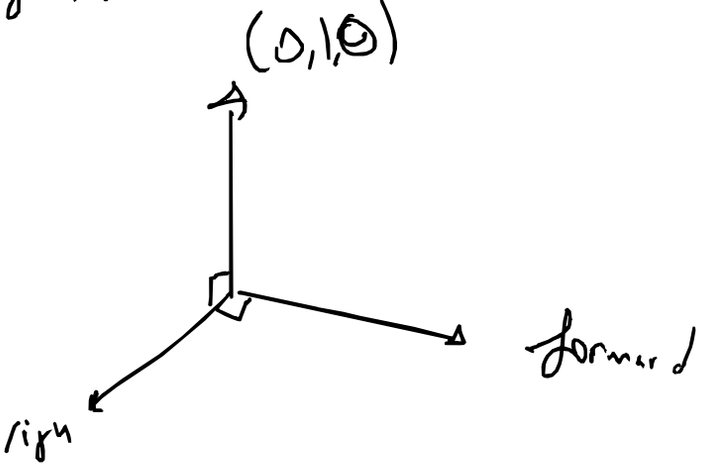
translation

$$\begin{bmatrix} & & & \end{bmatrix}$$

Centro con forward, por cámara.

Un Global up → (0, 1, 0).

Con eso: calculamos right.



Cross product define un vector perpendicular a ambos vectores.

Right ⇒ forward x up ✓ con regla del pulgar

Up verdadero ⇒ right x forward

eye, center.

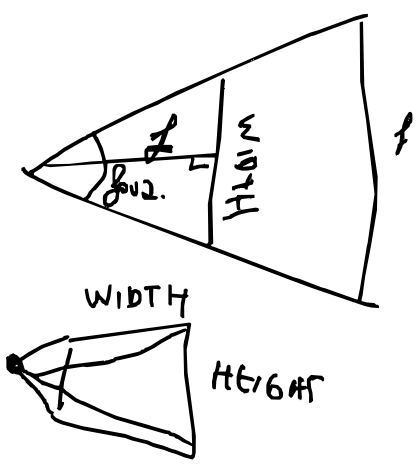
eye → donde está

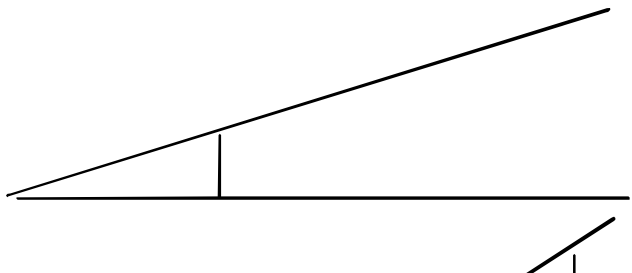
center → hacia donde mira.

$$\begin{bmatrix} right_x & right_y & right_z & dr \\ up_x & up_y & up_z & du \\ forward_x & f_y & f_z & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

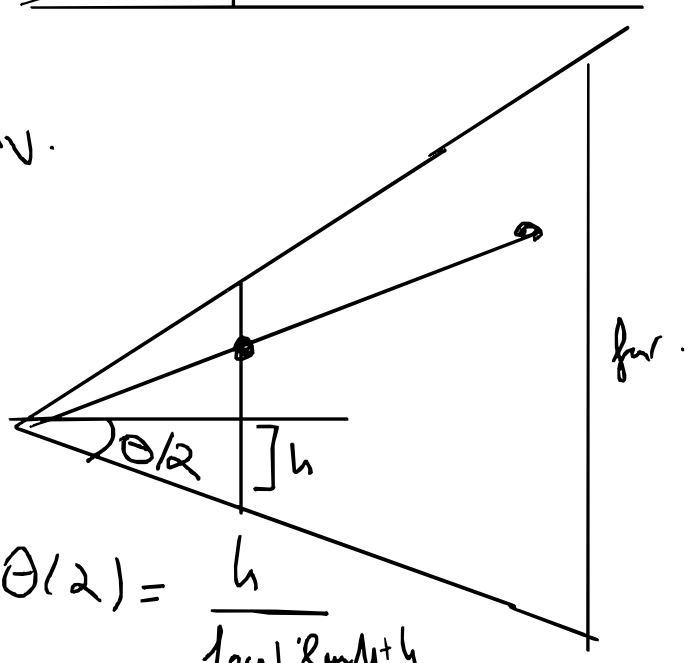
PROYECCIÓN.

$$fov = 60 .$$



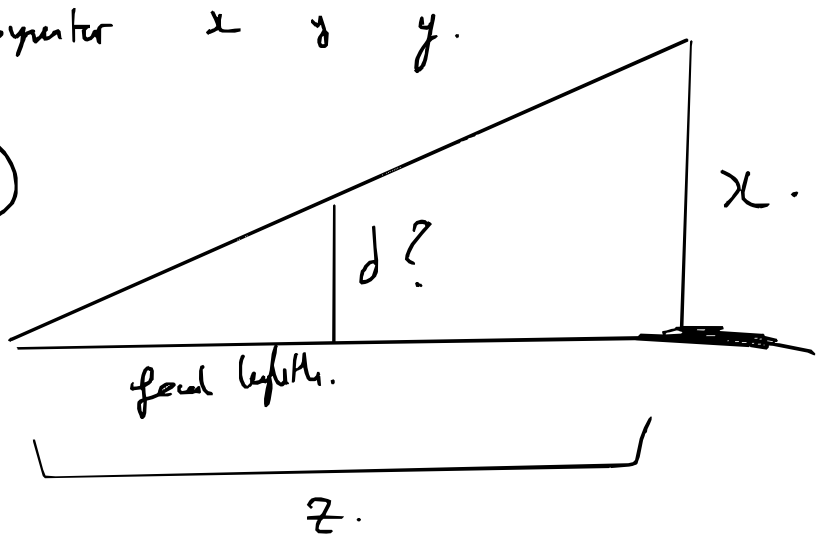


$\theta = fov.$



$$\tan(\theta/2) = \frac{h}{\text{focal length} + h}$$

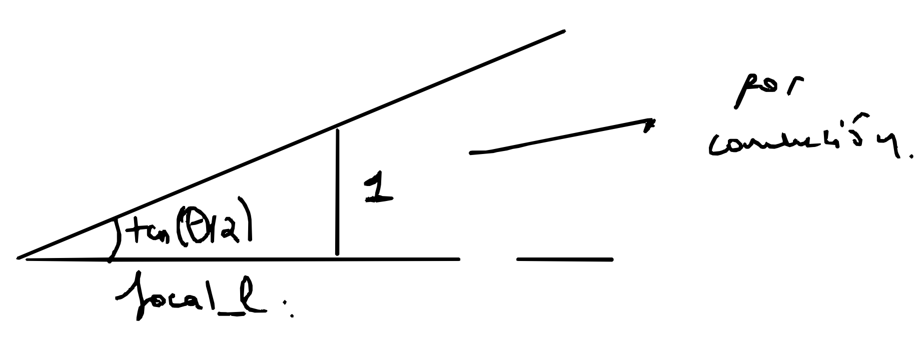
proyectar x y y.



$$\frac{d}{x} = \frac{\text{focal length}}{z}.$$

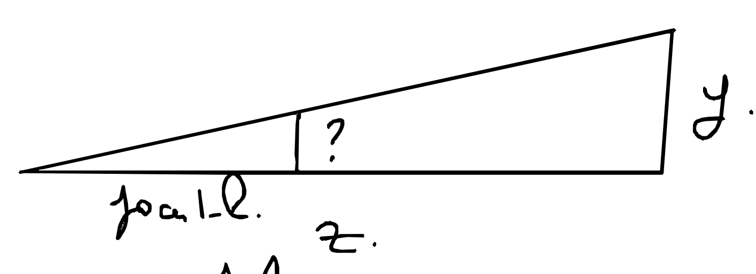
$$d = x \times \frac{\text{focal length}}{z}.$$

we first calculate focal length.



$$\tan(\theta/2) = \frac{1}{focal\_l}$$

$$\Rightarrow focal\_l = \frac{1}{\tan(\theta/2)}.$$



$$\frac{?}{y} = \frac{focal\_l}{z}.$$

$$new\_y\_ndc = y\ real \times \frac{focal\ length}{(z)}$$

↳ toca hacerlo después porque en matriz we cannot.

$$new\ x\ ndc = x\ real \times \frac{focal\ length}{z}.$$

x también mide -1 a 1. pero la pantalla es más ancha específicamente el ASPECT RATIO. veces más ancha.
por eso el x casi proyectado
↳ Toca dividirlo por esto
↳ para poderlo después multiplicar

$$focal\ length = \frac{1}{\tan(\theta/2)}.$$

$$\Rightarrow \begin{bmatrix} \frac{focal\ length}{aspect} & 0 & 0 & 0 \\ 0 & focal\ length & 0 & 0 \\ 0 & 0 & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

Queremos NDC por profundidad. toca que near plane sea -1 y far plane sea 1.

$$\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} \times \begin{bmatrix} \frac{f}{asp} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & m_1 & m_2 \\ 0 & 0 & 1 & 0 \end{bmatrix} \Rightarrow \begin{pmatrix} x_{casi\ proy.} \\ y_{casi\ proy.} \\ z \\ z \end{pmatrix}$$

↳ el problema tocará dividir por z, así que necesitamos conservar ese z. en z^2.

$$\begin{pmatrix} x\ casi\ proy \\ x\ casi\ proy \\ z^2 \\ z \end{pmatrix}.$$

tenemos $m_1 z + m_2 = z^2$.
↳ ecuación cuadrática
⇓
↳ solo hay dos soluciones max.

y queremos por NDC que -1 = sea near plane z y 1 sea far plane pero esto es diferente.

Para mapear near a -1 y far a +1 toca hacer lo siguiente:

$$\begin{bmatrix} \frac{f}{a} & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x\ clip \\ y\ clip. \\ z\ clip. \\ z \end{bmatrix}$$

sabemos que para $z_{ndc} = \frac{z\ clip}{z}$.

$z\ clip = Az + B$.

$$z\ clip = A z_{real} + B$$

$$z_{ndc} = \frac{A z_{real} + B}{z}.$$

lo mismo que near.

$$Si\ z_{ndc} = -1 \Rightarrow -1 = \frac{A z_{real} + B}{near}.$$

$$-1 = A + \frac{B}{near}.$$

$$1 = A + \frac{B}{far}$$

$$\frac{B}{near} - \frac{B}{far} = -2.$$

$$\frac{B far - B near}{near \cdot far} = -2.$$

$$B far - B near = -2 \cdot (near \cdot far)$$

$$B(far - near) = -2 \cdot (near \cdot far).$$

$$B = -2 \cdot \frac{(near \times far)}{(far - near)}.$$

$$B = 2 \cdot \frac{(near \cdot far)}{(near - far)}.$$

$$-1 = A + \frac{B}{near}.$$

$$1 = A + \frac{B}{far}$$

$$-1 = A + \frac{2(near \times far)}{(near - far)(near)}.$$

$$-A = 1 + \frac{2(near \times far)}{(near - far)(near)}.$$

$$-A = \frac{(near - far)(near) + 2(near \times far)}{(near - far)(near)}.$$

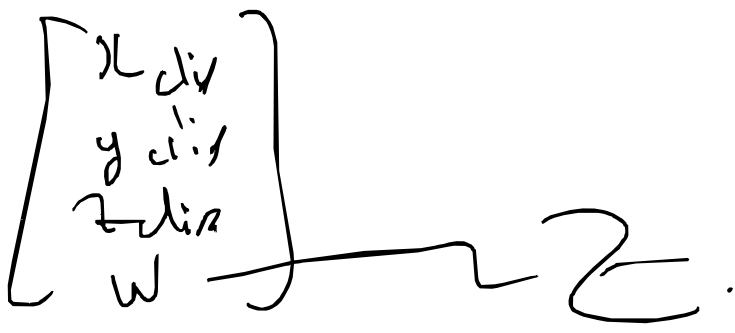
$$-A = \frac{(near - far) + 2 far}{near - far}$$

$$-A = \frac{near + far}{near - far}$$

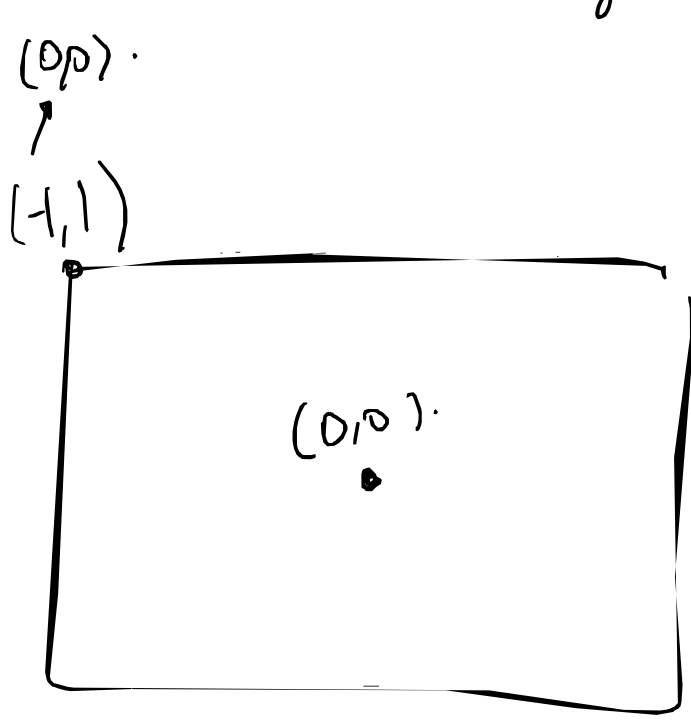
$$A = \frac{near + far}{far - near}.$$

MVP = rot x . rot y . rot z .

- scale . translation .
- look at
- projection.



NDC $\in$. x, y .

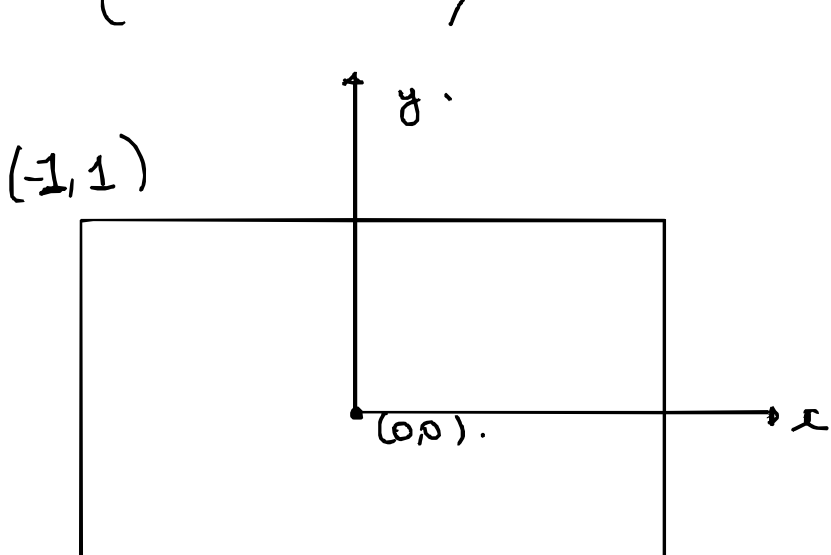


$(-1,1) \to (0,0)$

$(0,0) \to \left(\frac{WIDTH}{2}, \frac{HEIGHT}{2}\right)$.

$(1,1) \to (WIDTH, HEIGHT)$.

$(1,1) \to (WIDTH-1, HEIGHT-1)$

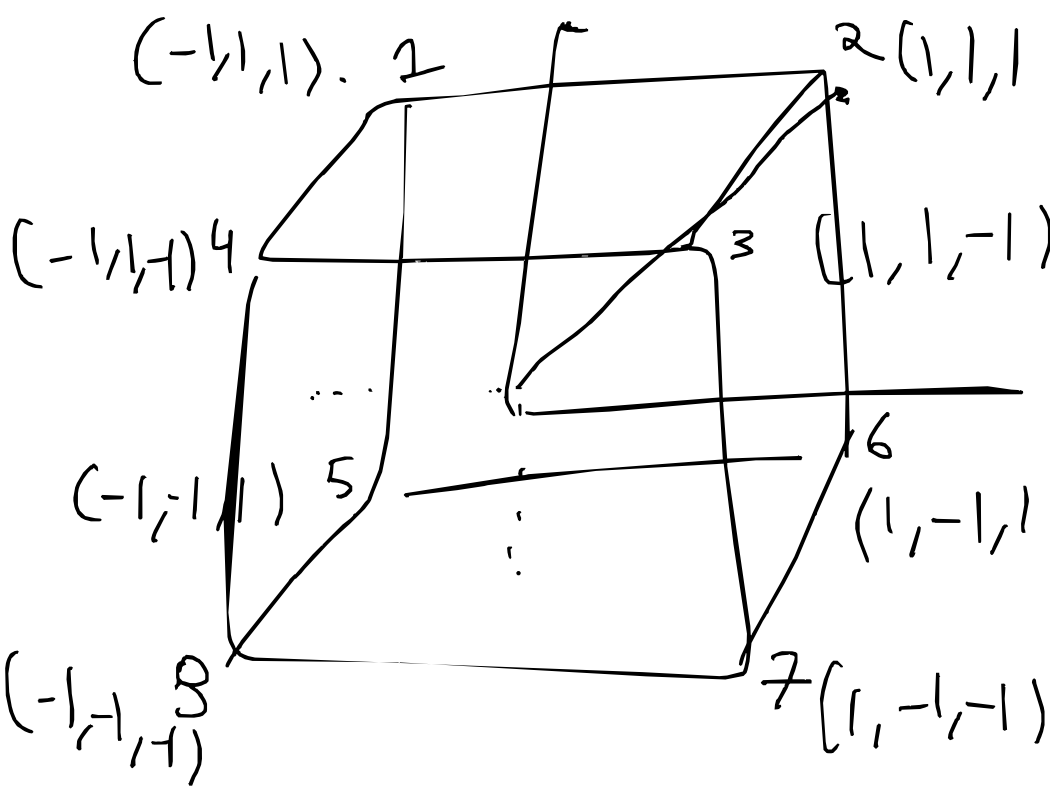


transformations a (0,2) ⇒ 0

y a (2,0)

$\Rightarrow (ndc\,x + 1) \times \frac{width - 1}{2}$

$\Rightarrow (1 - ndc\,y) . x \frac{height - 1}{2}$.



1 2 3
1 3 4

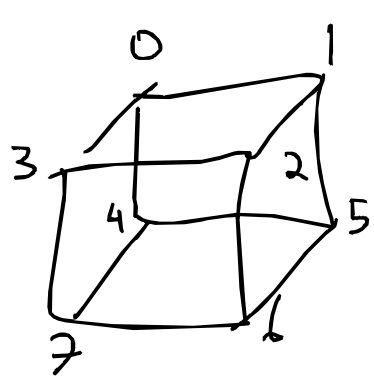
5 6 7
5 7 8

1 2 6
1 6 5

4 3 7
4 7 8

1 4 8
1 8 5

2 3 7
2 7 6



18 19 20 21 → width.

22 23 24 25 → height.

for height − 1

for int width * 3.

para llegar al pixel anterior

de 4.

(width x3)

39 → 40 ⇒ 1

42 → 44. ⇒ 2.

41 → 44 ⇒ 3

40 → 40 ⇒ 0

(4 − ((width *3) % 4)).

1 → 4 ⇒ 3

2 → 4 ⇒ 2

3 → 4 ⇒ 1.

4 → 4 ⇒ 0

5 → 8 ⇒ 3

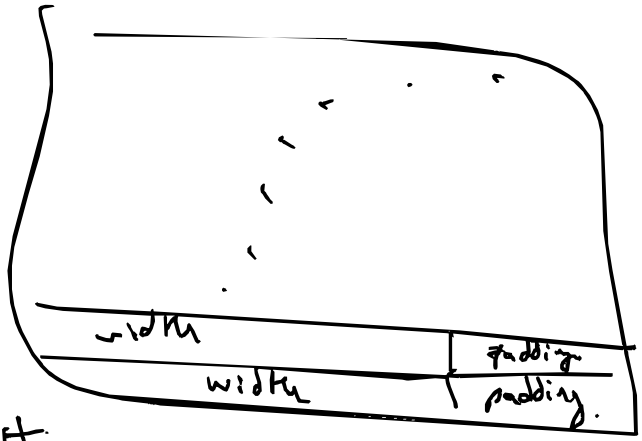
6 → 8 ⇒ 2.

7 → 8 ⇒ 1

int resta, = int

	mod 4.	numero anterior	resta.
1	1	4.	3
2	2	4	2
3	3	4	1
4	0	4	0
5	1	8	3
6	2	8	2
7	3	8	1
8	0	8	0.

⇒ (4 − ((width*3) % 4)) % 4.



Start

+ struct faces.

Class Mesh.

- vertices.
- uvs.
- normals.
- faces → face1 — v_indices
  - uv_indices
  - n_indices.
  - material_name.
- materiales.

+ Load o!

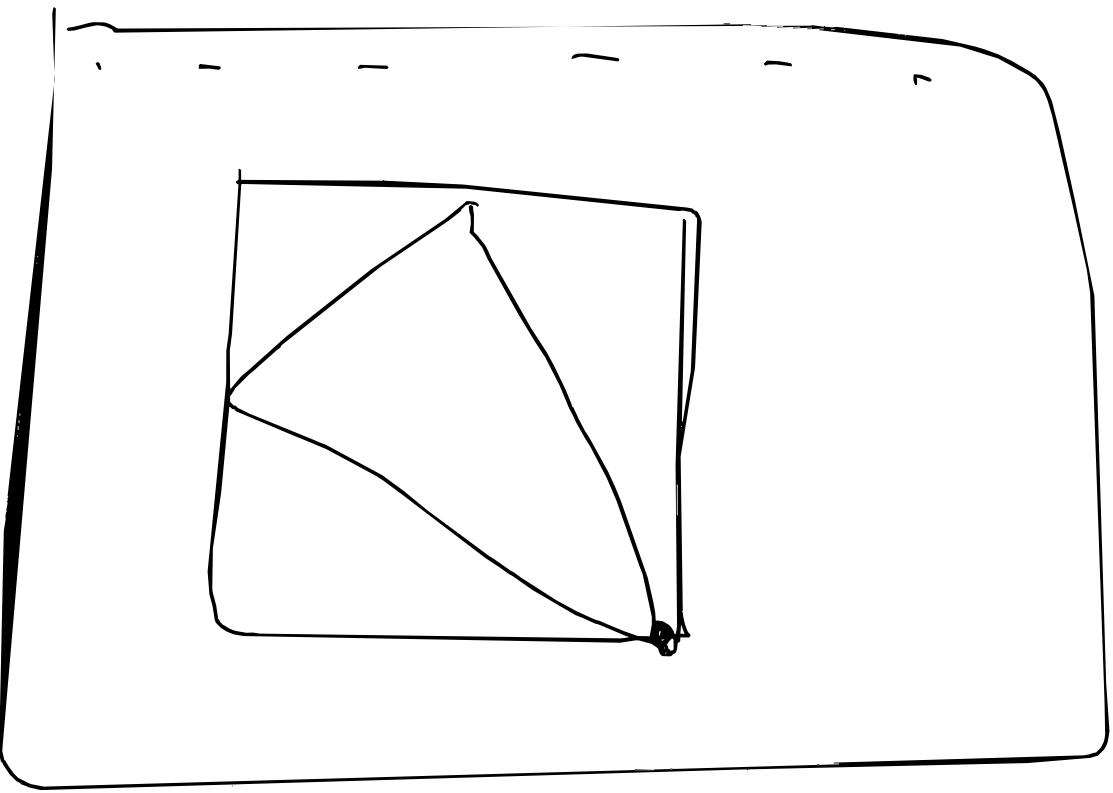
Class Material.

- numbr
- texture.

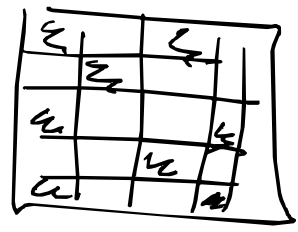
+ method → load texture.

Class render

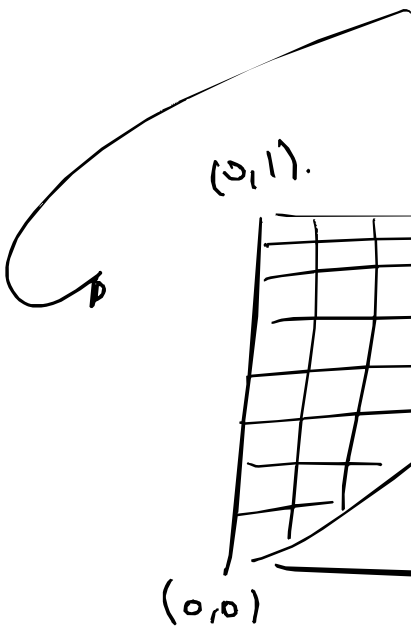
- Rasterize Hiyple.
- draw mesh



Rasterizer ←



p1, p2, p3
uv pontos. del triangulo.



eABnorm x uvC + eBC x uvA + eAC x uvB

sin depth correct interpolation.

y despues multiplicamos

uv (x, y).

pix_x = (tex_width+1) x uv_x.

pix_y = (tex_height+1) x (1 − uv_y).

face. = v_in { 0 1 2 3 − , − , − , −

v_n { − , − , − , −

v_t. { − , − , − , −

0 1 2
0 1 3

0 i i+1. si son 5.
0 i i+2.

1 2 3

1 3 4

1 4 5

Boy → Ruta de acceso no servía por working directory.

ruta mtl no servía porque por pendejo reiniciaba mtl.

reiniciaba el material encontrado en el while mientras leía líneas.

las direcciones en mtl estaban mal puestas de \ a "/" y de jpg a bmp.

nombre de archivo

dirt grass → dirt_grass.

no lee las faces, → sí lee. Como no hacía push back.

depth es igual a

$$\Rightarrow \text{edge}AB \times \left(\frac{1}{w_C}\right) + \text{edge}BC \times \left(\frac{1}{w_A}\right) + \text{edge}(AC) \times \left(\frac{1}{w_B}\right).$$

↳ esto es lineal en pantalla porque no hay profundidad.

$\frac{1}{w}$ es lineal.

y si vemos $\text{edge}AB \times \left(\frac{uv_C}{w_C}\right) + \text{edge}BC \times \left(\frac{uv_A}{w_A}\right) + \text{edge}AC \times \left(\frac{uv_B}{w_B}\right)$.

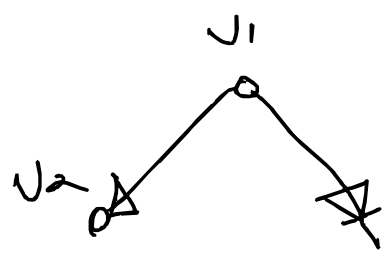
↳ es lineal también porque si vemos no hay profundidad.

$\frac{uv}{w}$ es lineal.

por lo tanto $uv = \frac{uv}{w} \times w = uv$.

por lo tanto

$$\frac{uv}{w} \times \frac{1}{\text{depth}} \Rightarrow uv.$$



cross de ambos ⊥ normal de cara.

* Working Directory.

→) Muy importante

Bug. memory access.

a veces → key frames

→ accede a memoria que no existe.

↳) no pintar cuando no se puede.

bad optimization → Debug + Release

↳ No hacer referencia de cosas como Face, uvs, uvs.

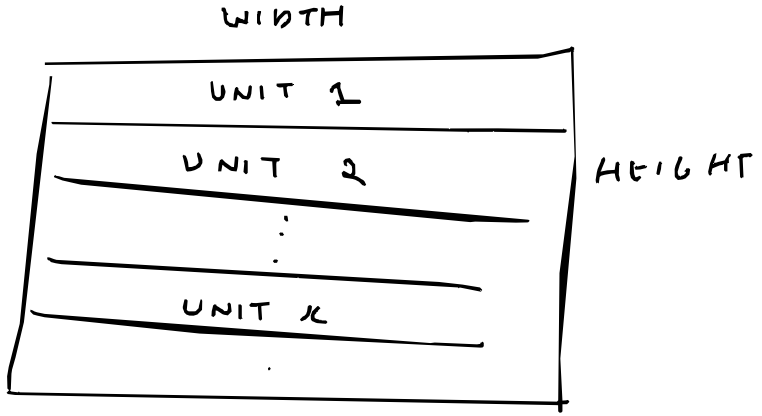
↳ Sobre todo, copiar material cada frame

MULTITHREADING.

HEIGHT

WIDTH

OF UNITS.



HEIGHT // UNITS. = Si entero, exito, sino.

HEIGHT = UNITS x FILAS por UNIT + Resto de Filas.

↳ Se la damos al último thread.

PHONG

$$\lambda = k_a I_a + k_d I_e (n \cdot l) + k_s I_e (n \cdot v)^n$$

$k_a \rightarrow$ constante ambiente

$k_d \rightarrow$ constante difusa

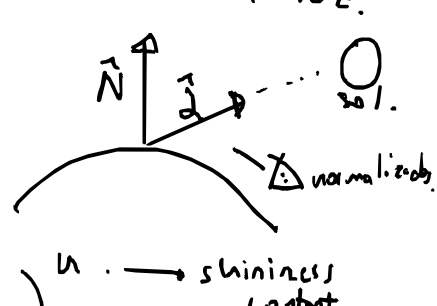
$k_s \rightarrow$ constante especular.

$$I_{intensidad\ Total} = I_{ambiente} + I_{difuse} + I_{specular}$$

$$I_{ambiente} = k_a \times I_A.$$

$$I_{difusa} = k_d \times I_2 (\vec{N} \cdot \vec{l})$$

Normal superficie × dirección hacia la luz.

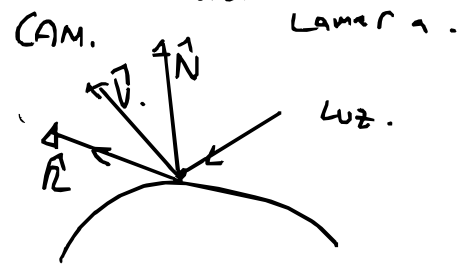


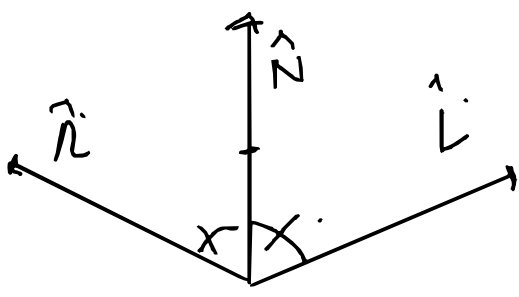
$$I_{specular} = k_s \times I_s (\hat{v} \cdot \hat{r})^n$$

$n \rightarrow$ shininess constant

$\hat{r} \rightarrow$ vector reflexion

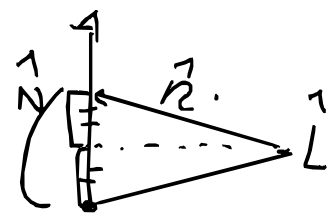
$\hat{v} \rightarrow$ vector hacia





$\vec{R} + \hat{L} = h\hat{N}$ ← obvio.

$2(\vec{L} \cdot \vec{N}) \Rightarrow$



$$\vec{R} = 2(\hat{L} \cdot \hat{N})\hat{N} - \hat{L}$$

$$pA = (x, y, z)$$

$$pB = (x, y, z)$$

$$pC = (x, y, z).$$

Coords 3D es:

$$e1\_norm * \left(\frac{pC}{depthC}\right) + e2\_norm * \left(\frac{pA}{depthA}\right) + e3\_norm * \left(\frac{pB}{depthB}\right).$$

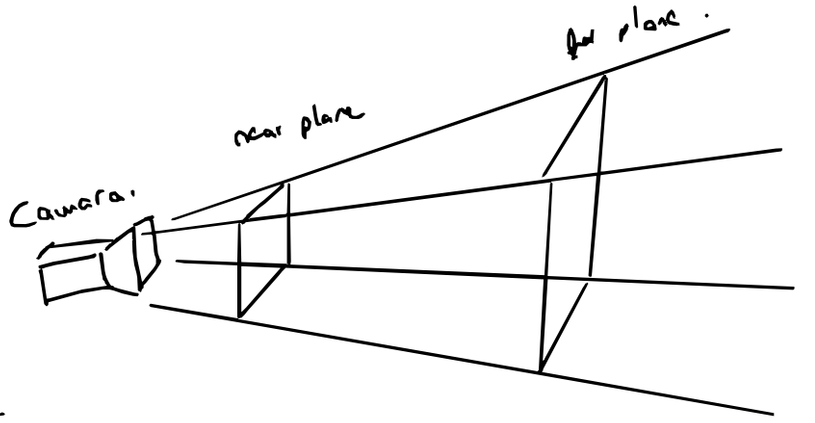
↳ multiplicamos por depth.

NS = shininess exponent.

Ka = constante amb

$Kd.$ = constante difusa

Ks = constante specular



↳ Coords NDC

necesitamos

near plane ⇒ -1

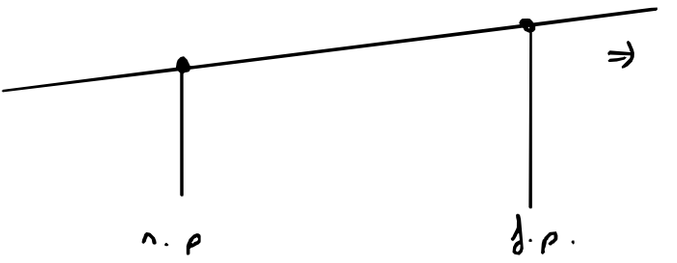
far plane ⇒ 1.

[-1, 1].

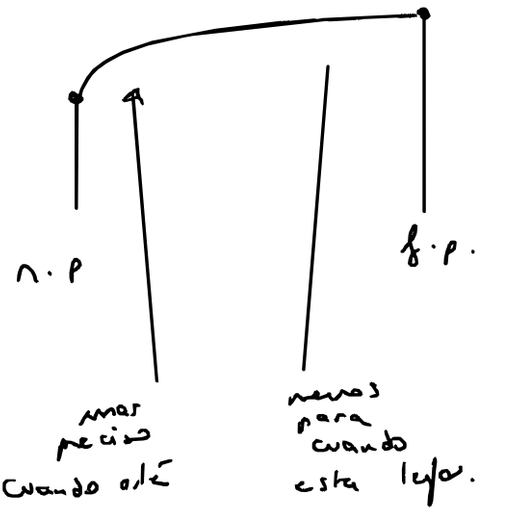
↳ lo visible por la camara.

↳ frustum culling.

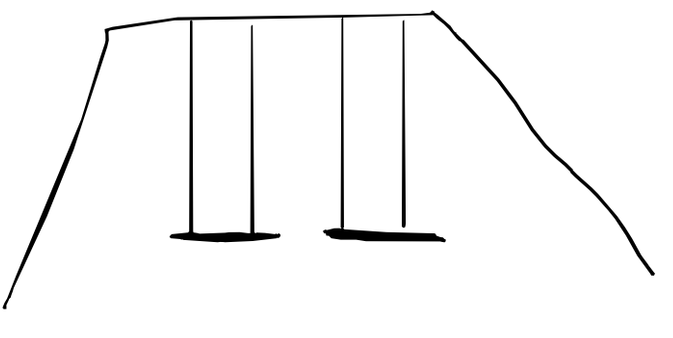
⇒ Problema, no se puede hacer linealmente porque la perspectiva no deforma uniformemente.



⇒ no se puede.



⇒ lo que hacemos.



TOON SHADING.

→ Black Outline. → dot punto a la cámara x normal de la cara

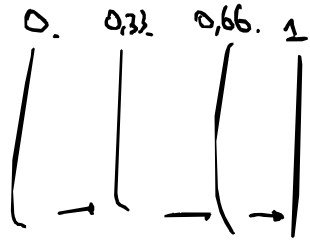
< 0.

→ Specular → 

Blanco.

 → dot reflectiva a la cámara.

↳ Toca hacer un clamp. a un valor cerca de ≈1 ⇒ convertir en 1.

→ Diffuse. → 

CAM.

pitch

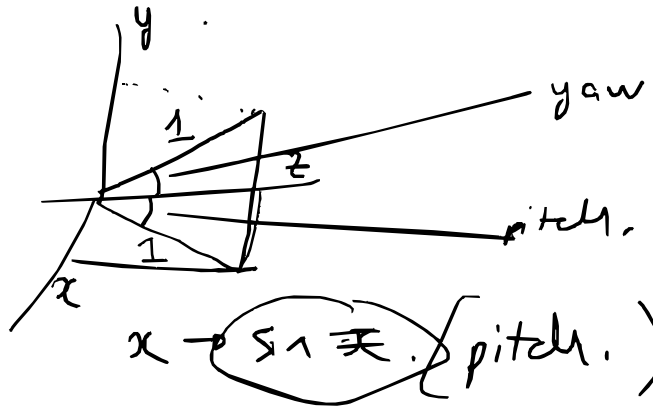
toca calcular con

pos camara.

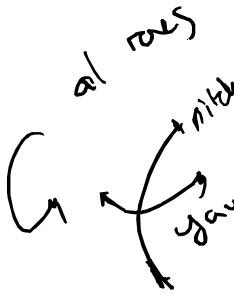
+ pitch y yaw

⇒ front up right

→ front.



x → sin z. (pitch.)



z → cos (pitch).

y → sin (yaw).

up I → 0, 1, 0 ∗ sin yaw. ∗ cos pitch

i sin pitch

right → cross z con yaw ∗ cos pitch

up. cross de right y front.

⇒ Coordenadas esféricas.

w a s d.

↓ ↓ ↓ ↓

pos pos pos pos

+ front − front + right − right.

Ahora. La luz.

Z buffer pero para la luz

⇒ Shadow Map.