



# 3D Without an Engine

By Trevor McCauley  
(aka senocular)



# 3D Without an Engine

Easy Peasy:  
“3D” Through **Scaling**

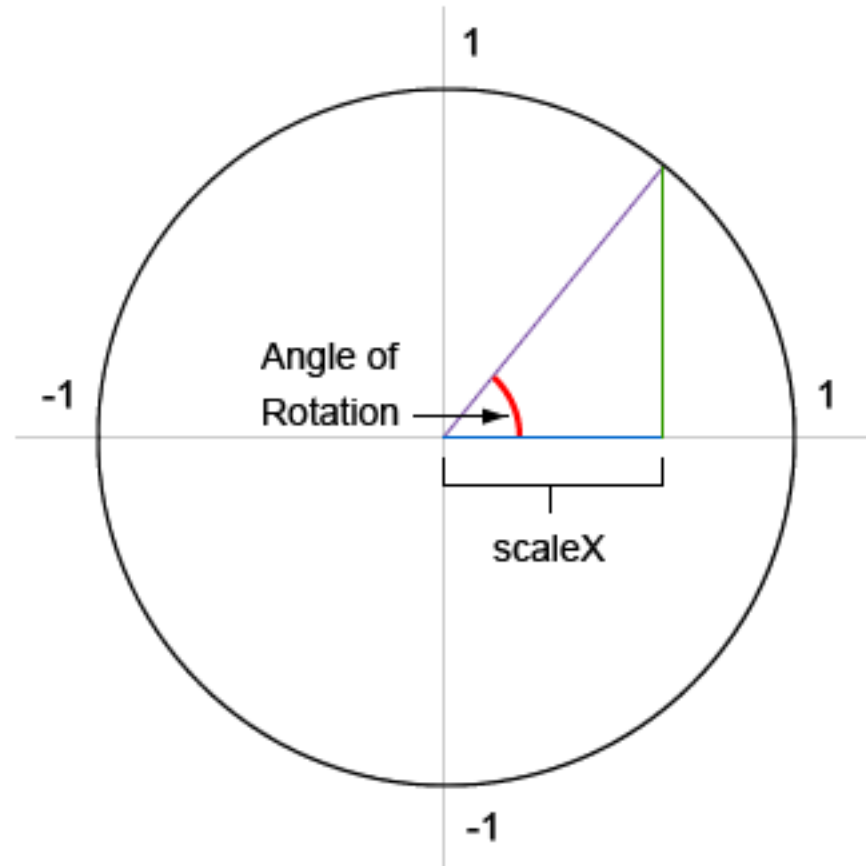


# 3D Without an Engine





# 3D Without an Engine





# 3D Without an Engine


$$E = mc^2$$

`scaleX = Math.cos(angle)`



# 3D Without an Engine

## Orthogonal Projection



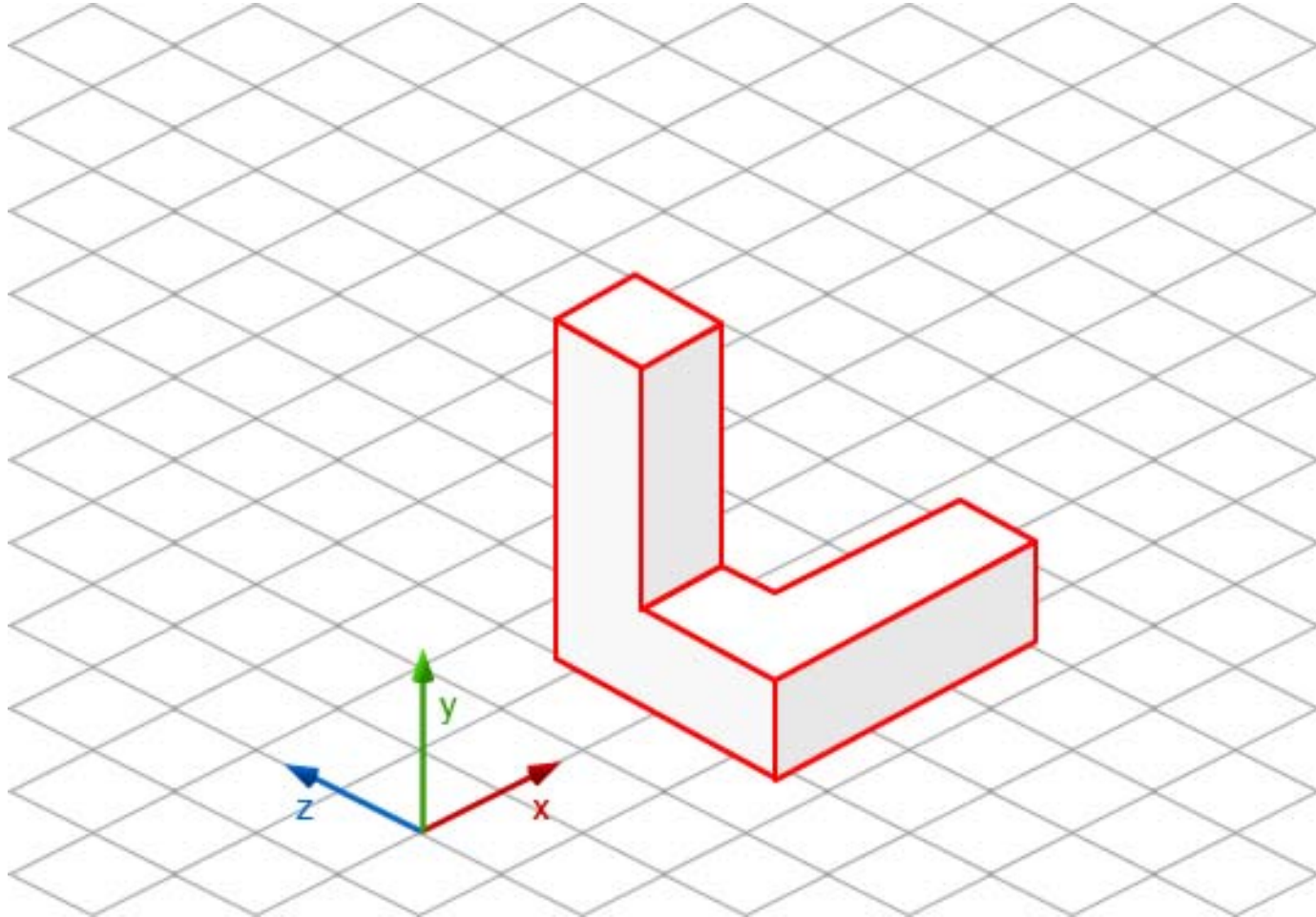
# 3D Without an Engine

Orthogonal Projection

**Isometric**

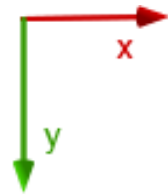


# 3D Without an Engine

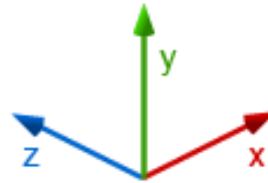




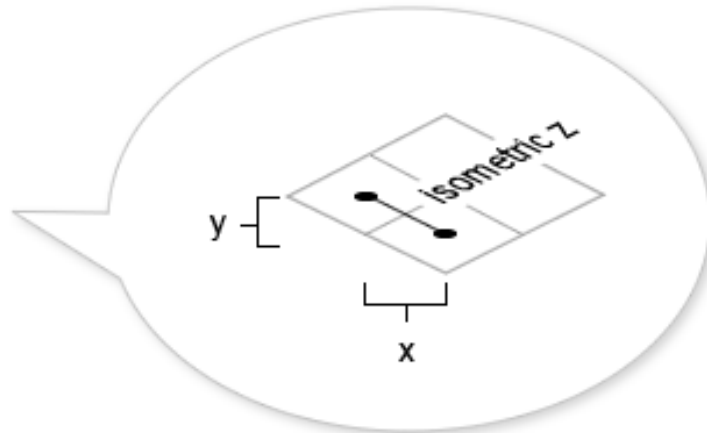
# 3D Without an Engine



TO



isometric  $x = x, -1/2 y$   
 isometric  $y = -y$   
 isometric  $z = -x, -1/2 y$





# 3D Without an Engine



x = isometric x - isometric z

&

y = -isometric x/2 - isometric z/2 - isometric y



# 3D Without an Engine

No perspective in  
isometric views





# 3D Without an Engine

Now you have to think:  
Scaling with **Perspective**



# 3D Without an Engine

**Near**

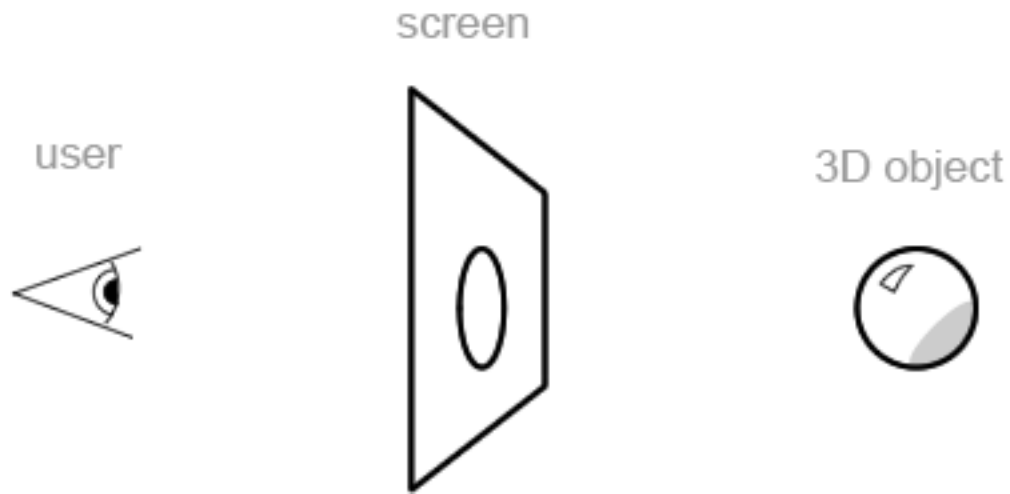


**Far**





# 3D Without an Engine

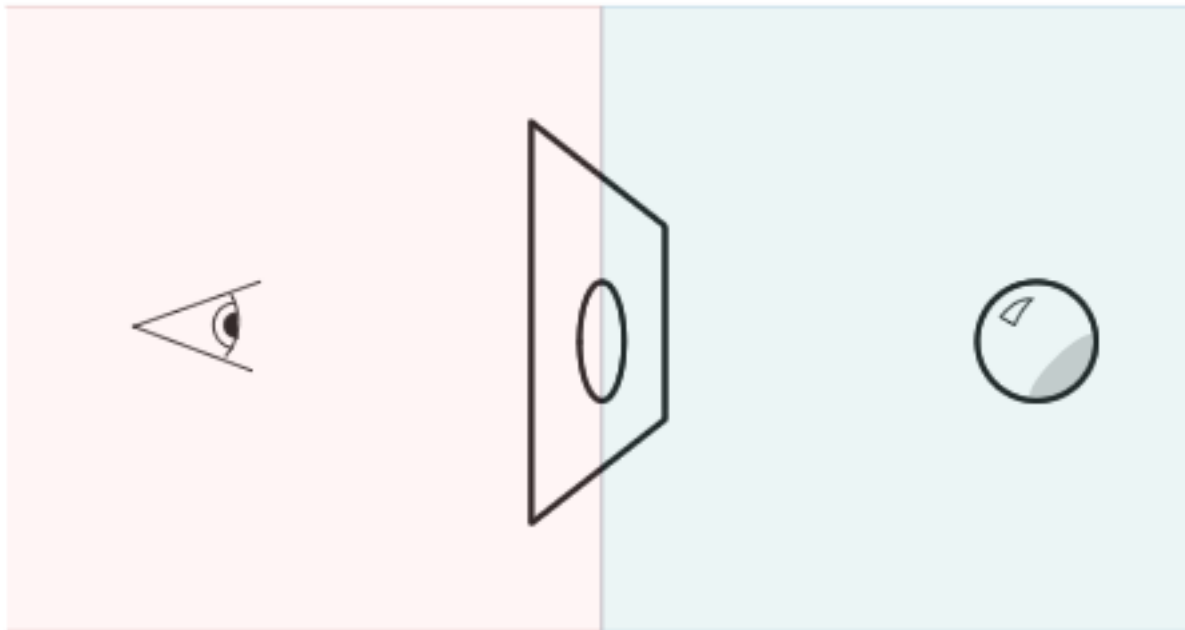




# 3D Without an Engine

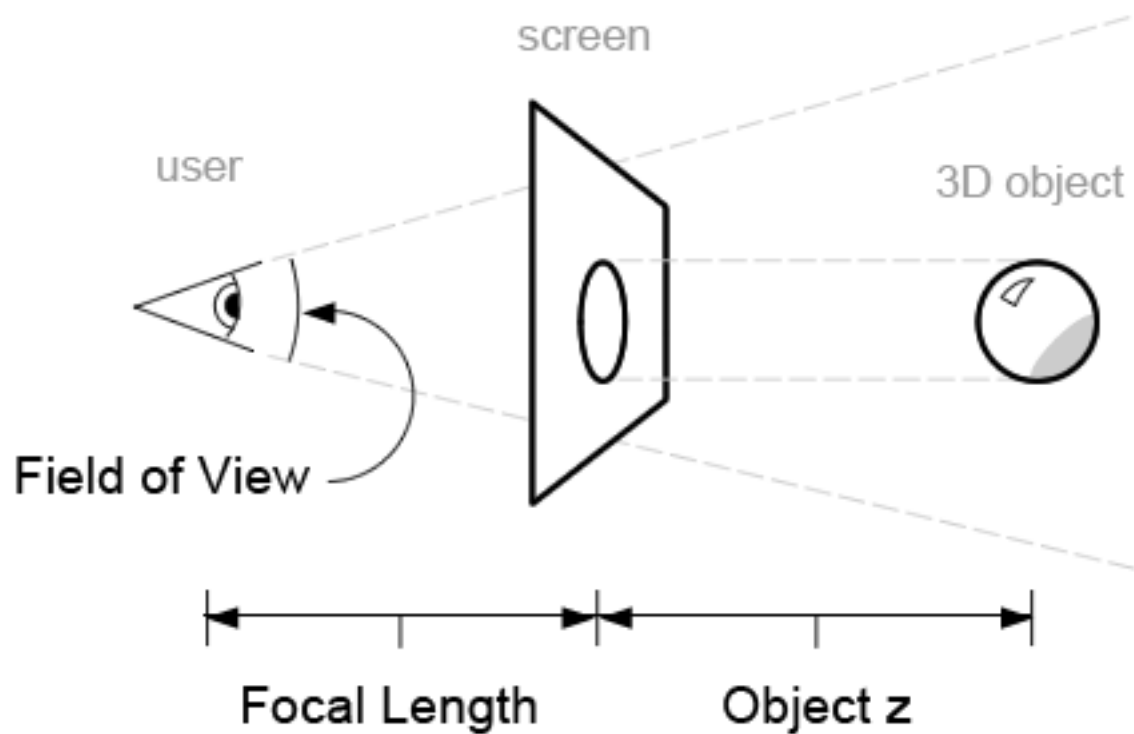
Perceived Reality

Imaginary 3D Space



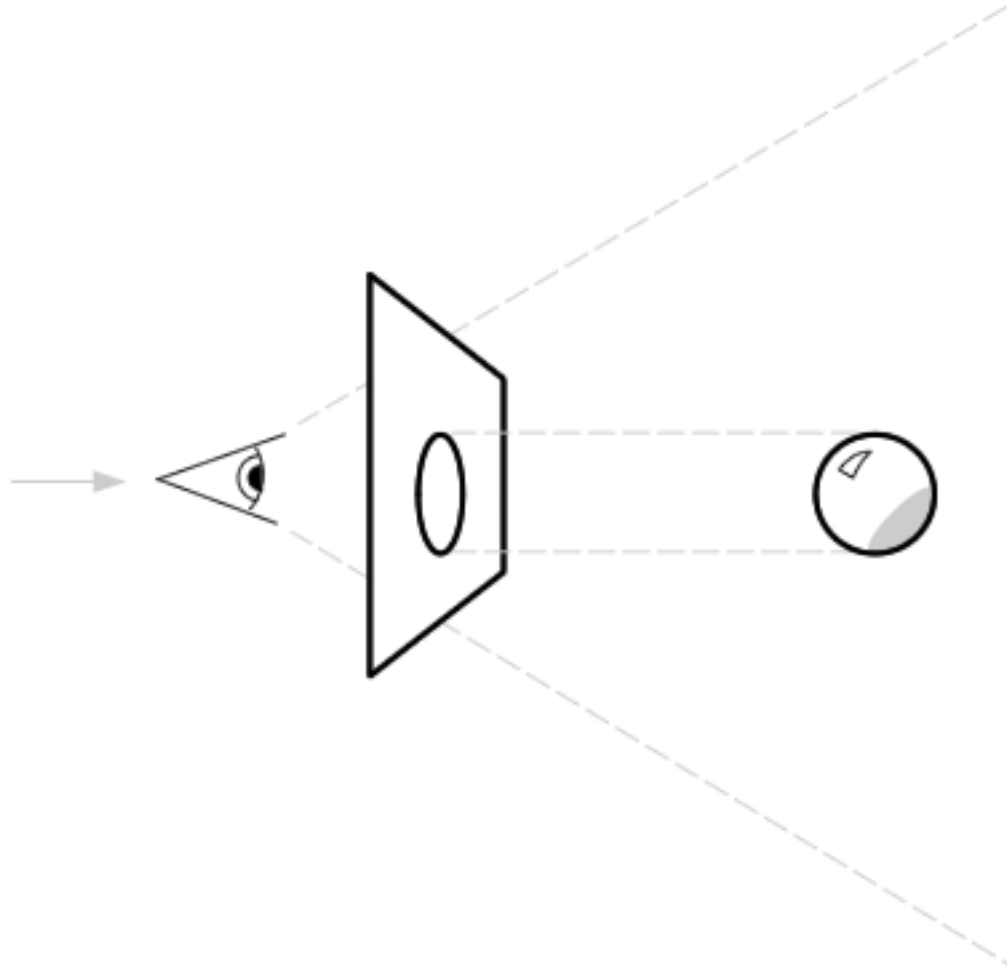


# 3D Without an Engine



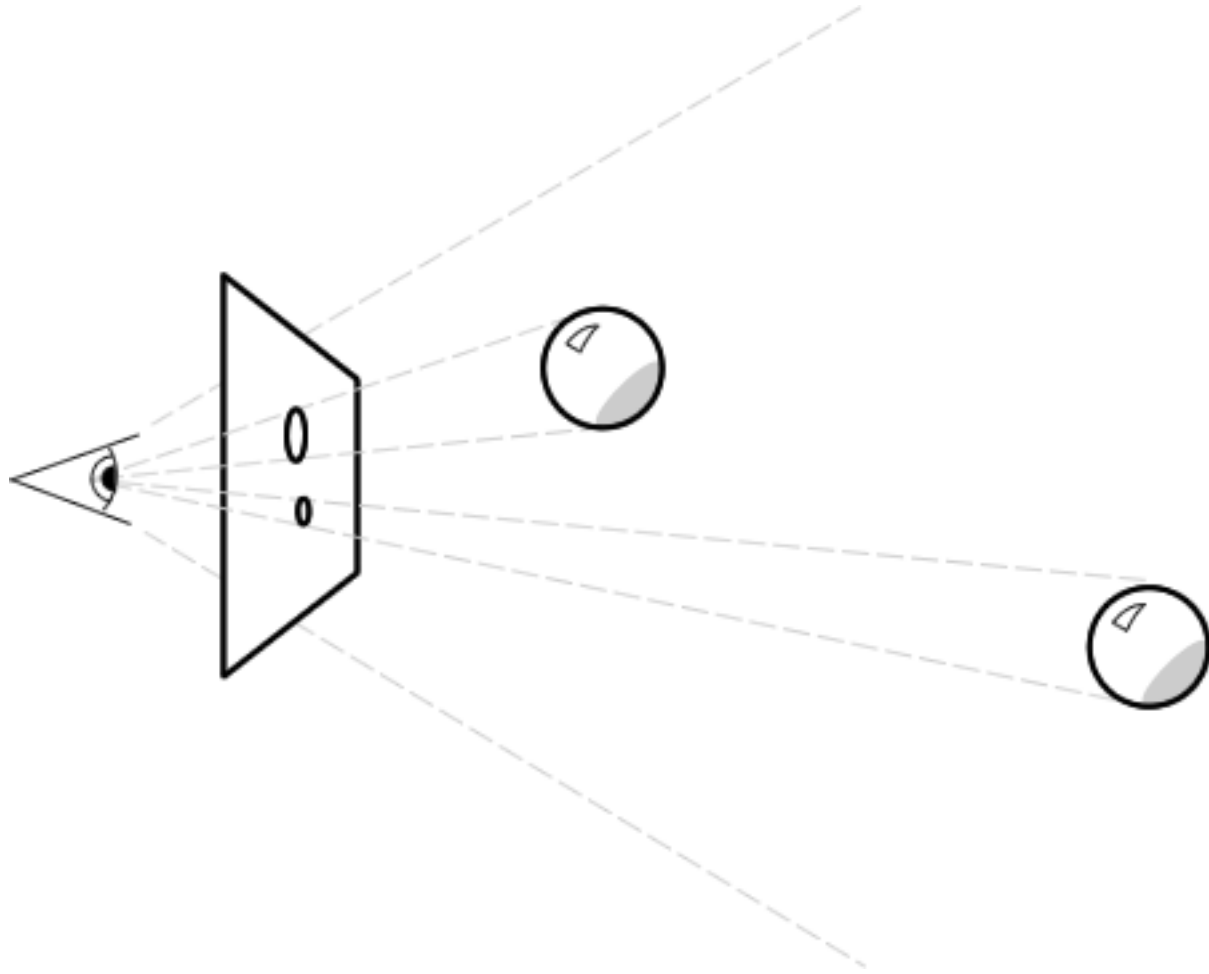


# 3D Without an Engine





# 3D Without an Engine





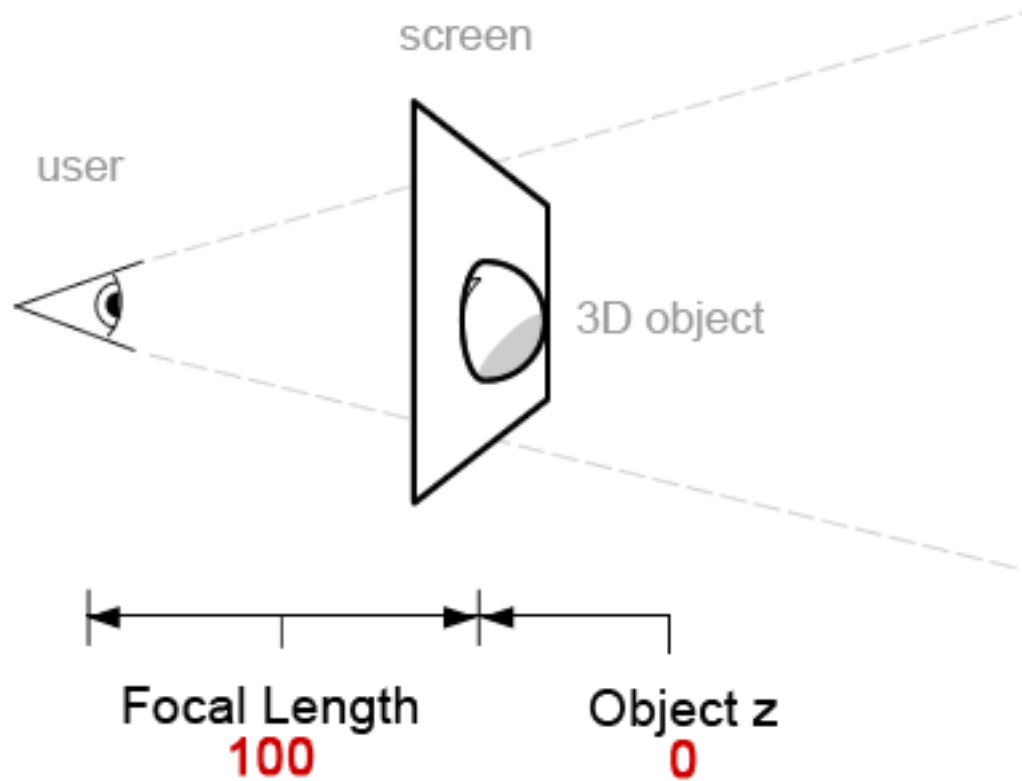
# 3D Without an Engine

$$e = mc^2$$

proportion = focal length / (focal length + z)



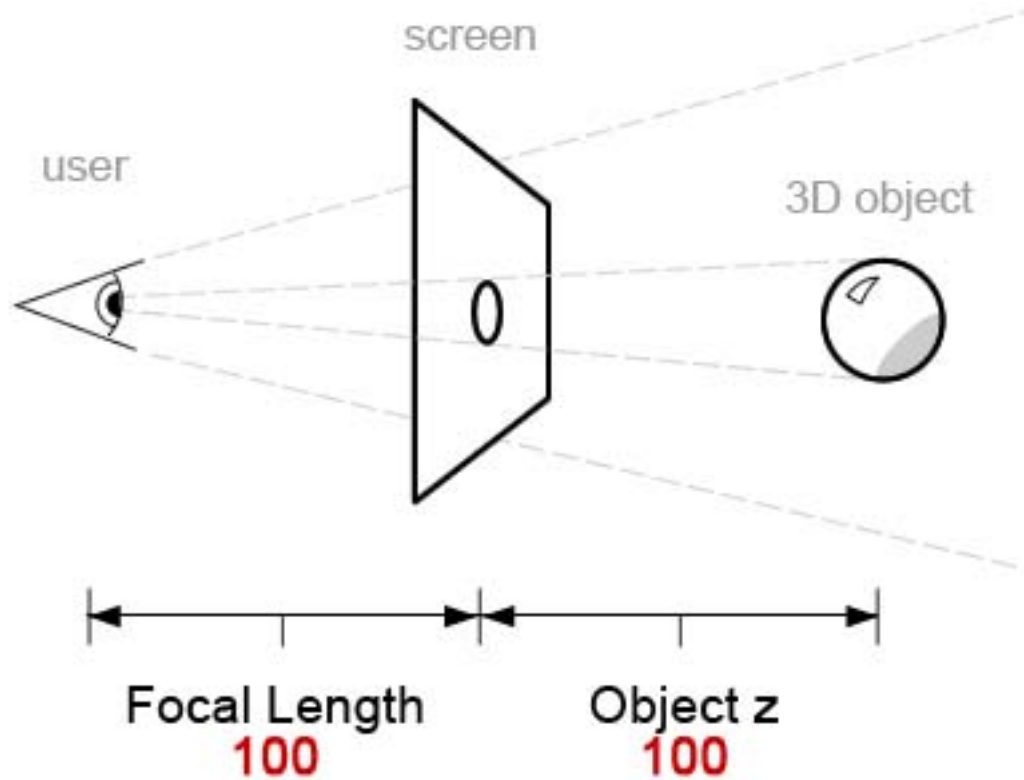
# 3D Without an Engine



$$\text{proportion} = 100 / (100 + 0) = \boxed{1}$$



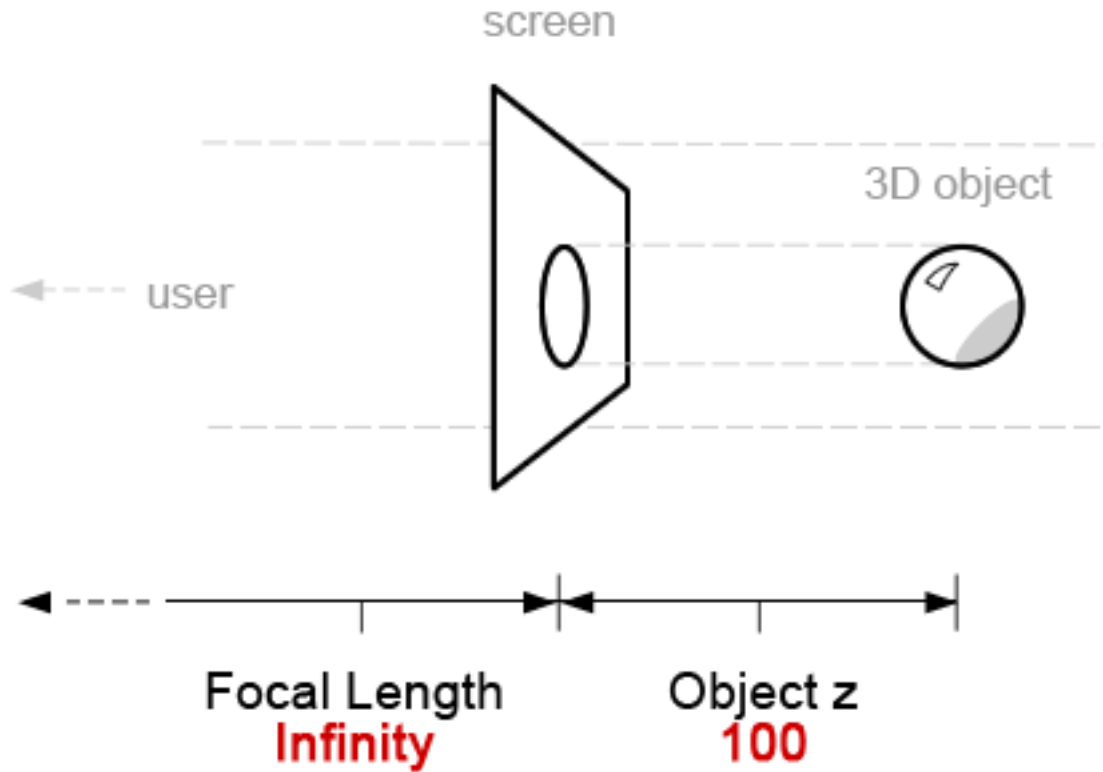
# 3D Without an Engine



$$\text{proportion} = 100 / (100 + 100) = \boxed{1/2}$$



# 3D Without an Engine



$$\text{proportion} = \text{infinity}/(\text{infinity}+100) = \boxed{1}$$



# 3D Without an Engine

$$e = mc^2$$

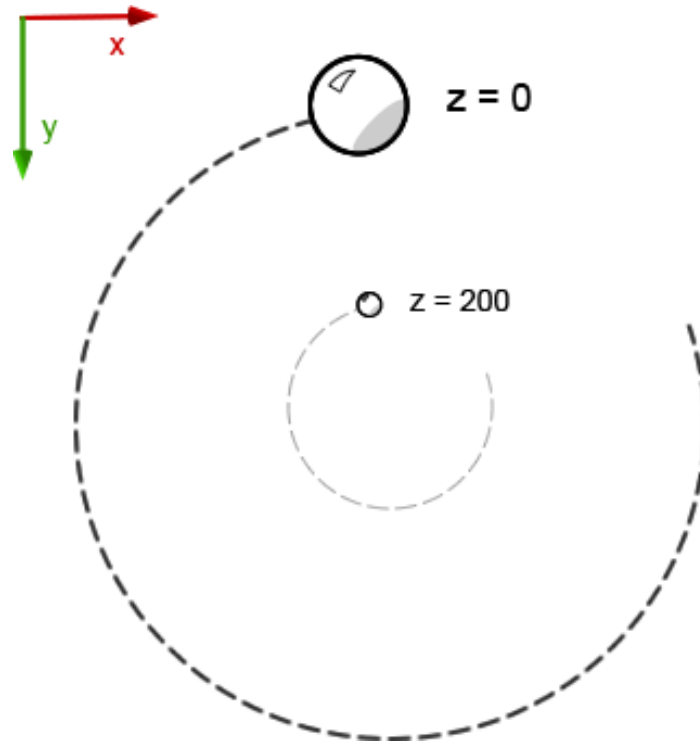
proportion = focal length / (focal length + z)

**scaleX = proportion**

**scaleY = proportion**



# 3D Without an Engine





# 3D Without an Engine

proportion = focal length / (focal length + z)

scaleX = proportion

scaleY = proportion

$x(2D) = \text{proportion} * x(3D)$

$y(2D) = \text{proportion} * y(3D)$



# 3D Without an Engine

When thinking starts to hurt:  
Proportional Bitmap Scaling  
**(Textures)**



# 3D Without an Engine





# 3D Without an Engine

Scale



Skew





# 3D Without an Engine



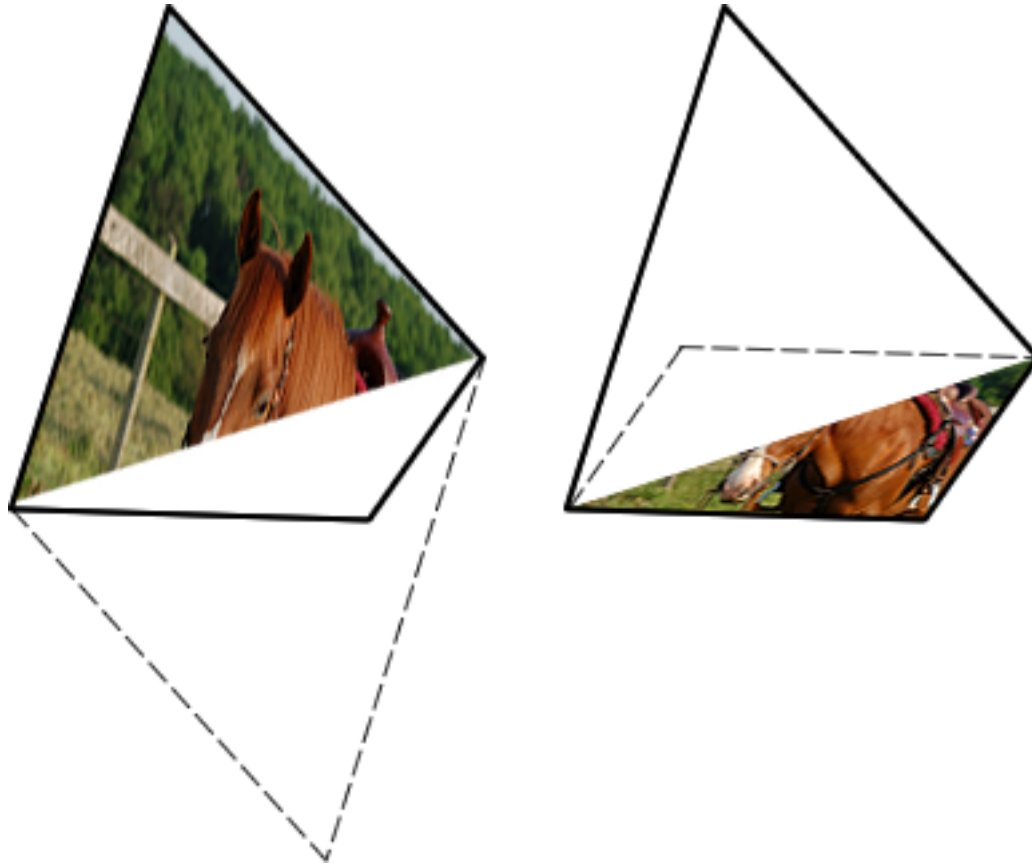


# 3D Without an Engine





# 3D Without an Engine





# 3D Without an Engine





# 3D Without an Engine

More triangles = greater precision



# 3D Without an Engine





# 3D Without an Engine





# 3D Without an Engine





# 3D Without an Engine





# 3D Without an Engine



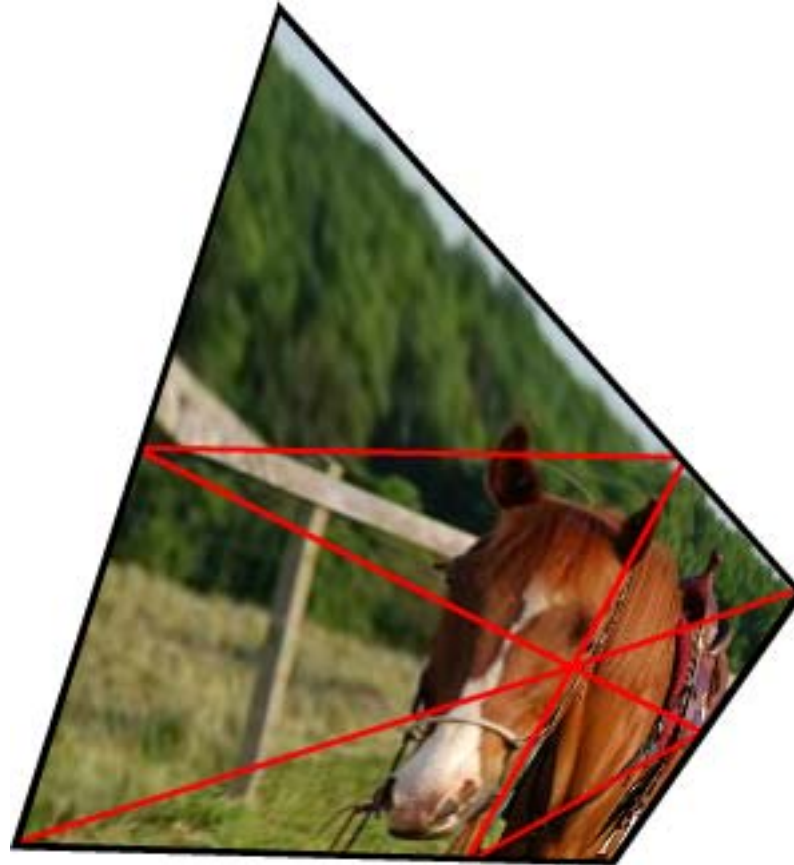


# 3D Without an Engine



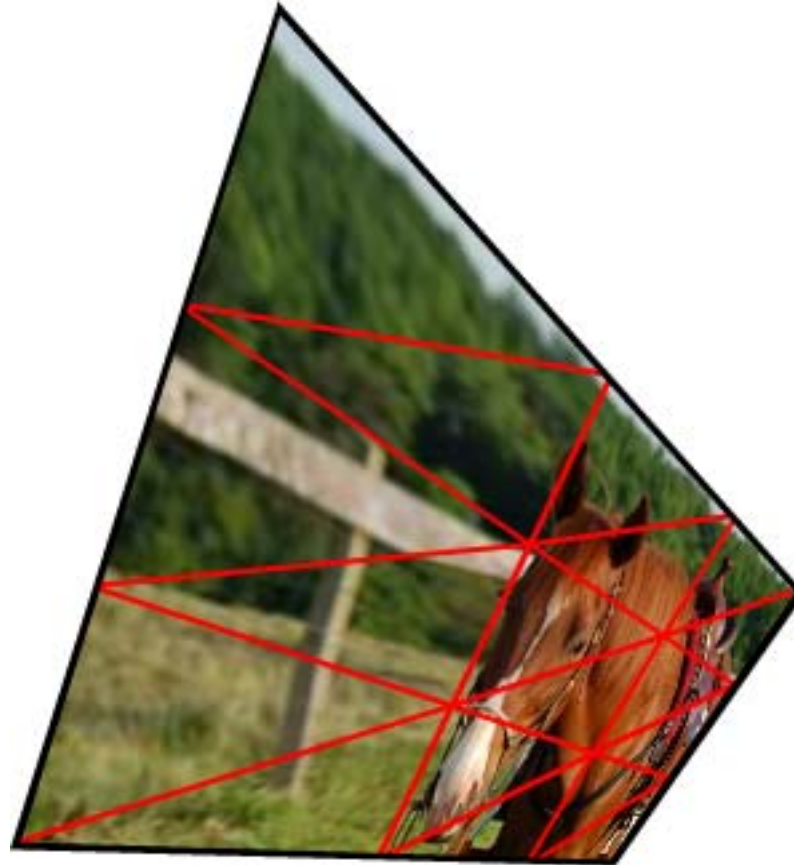


# 3D Without an Engine



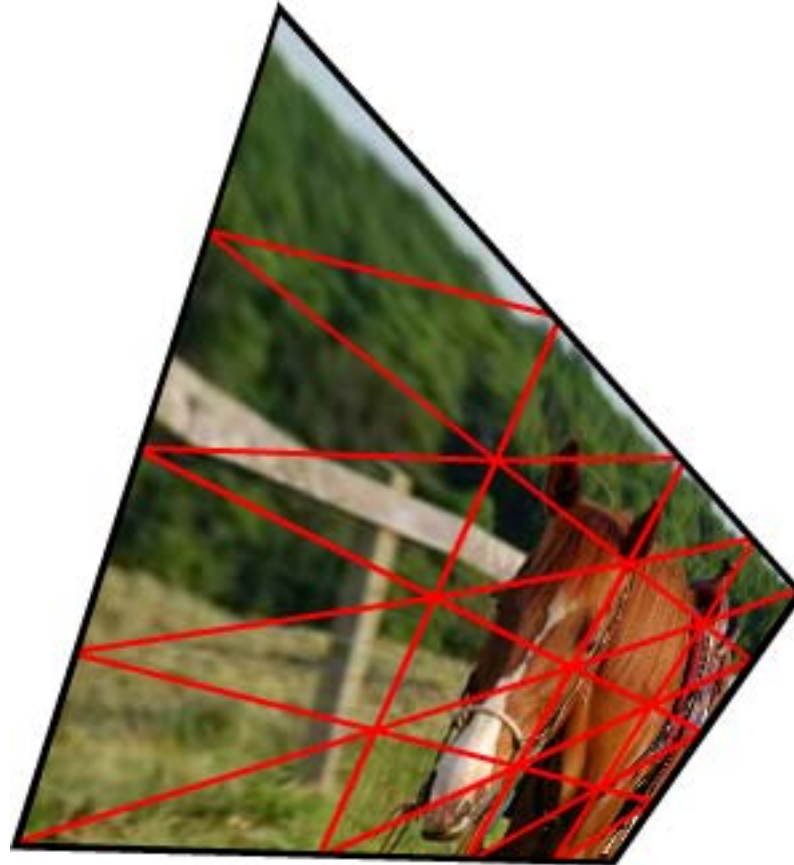


# 3D Without an Engine



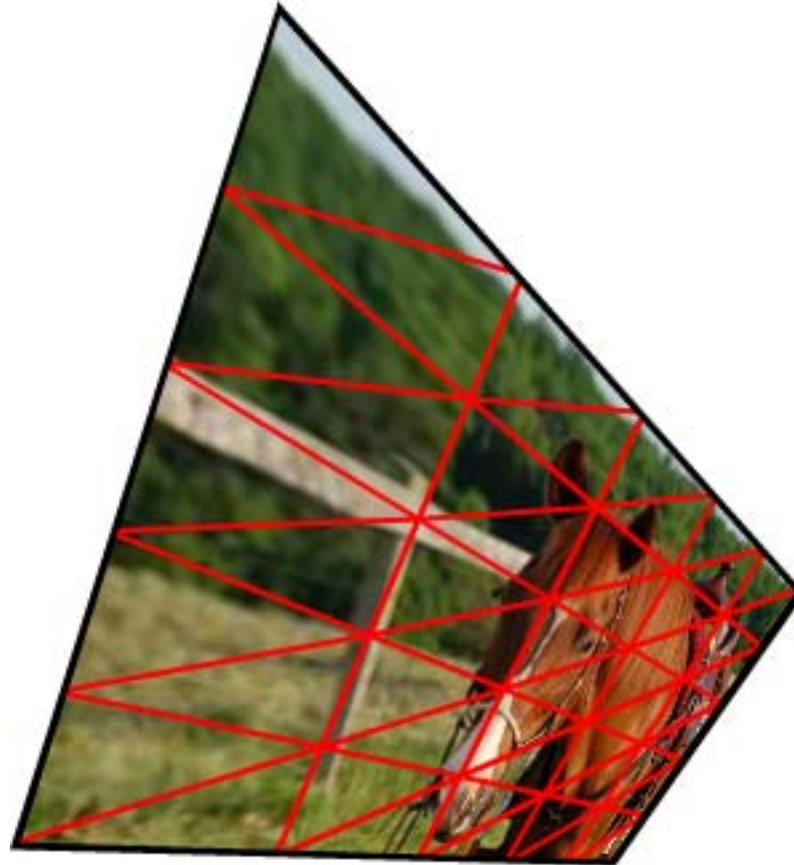


# 3D Without an Engine





# 3D Without an Engine





# 3D Without an Engine

$$E = mc^2$$

**Matrix = ... something complicated...**

(3D engines like Papervision3D make it easier)



# 3D Without an Engine

Helping you to think less:  
Flash Player 10 and  
**drawTriangles()**



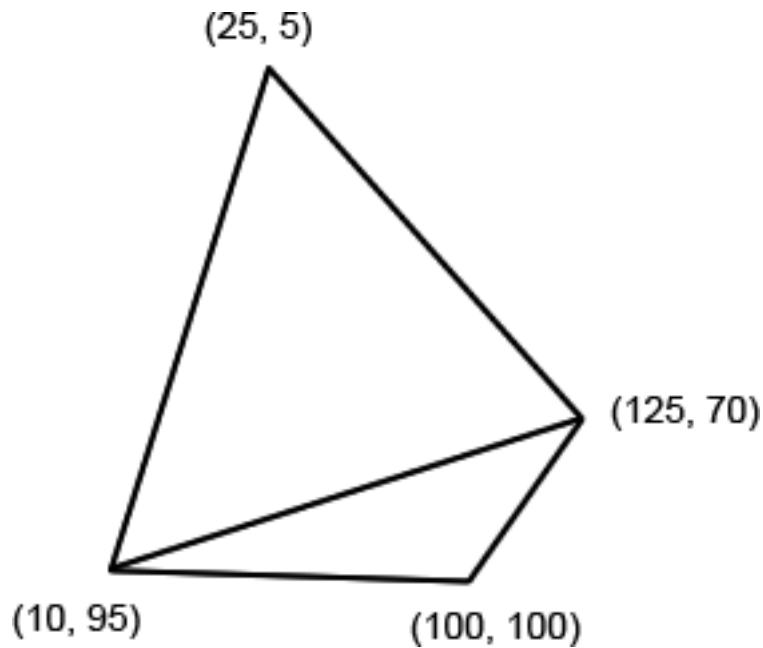
# 3D Without an Engine

## Graphics.drawTriangles(

- Vector of 2D coordinates (**vertices**),
- (Optional) Vector of **indices** for those vertices,
- (Optional) **UV** coordinates to map vertices to a bitmap with optional **T** for proportion,
- (Optional) **culling**);



# 3D Without an Engine

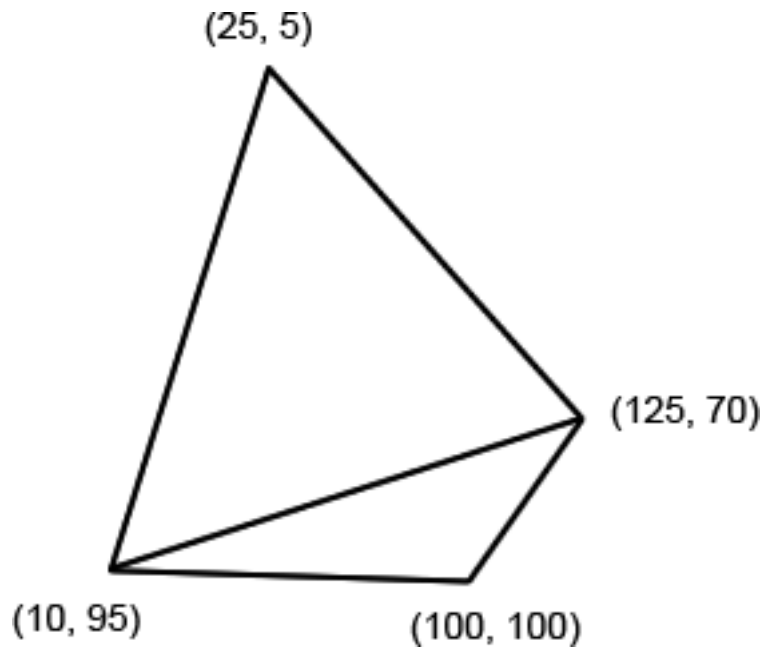


- **Vertices:**

- 25,5,
- 125,70,
- 10, 95,
- 125, 70,
- 100,100,
- 10,95



# 3D Without an Engine



- **Vertices:**

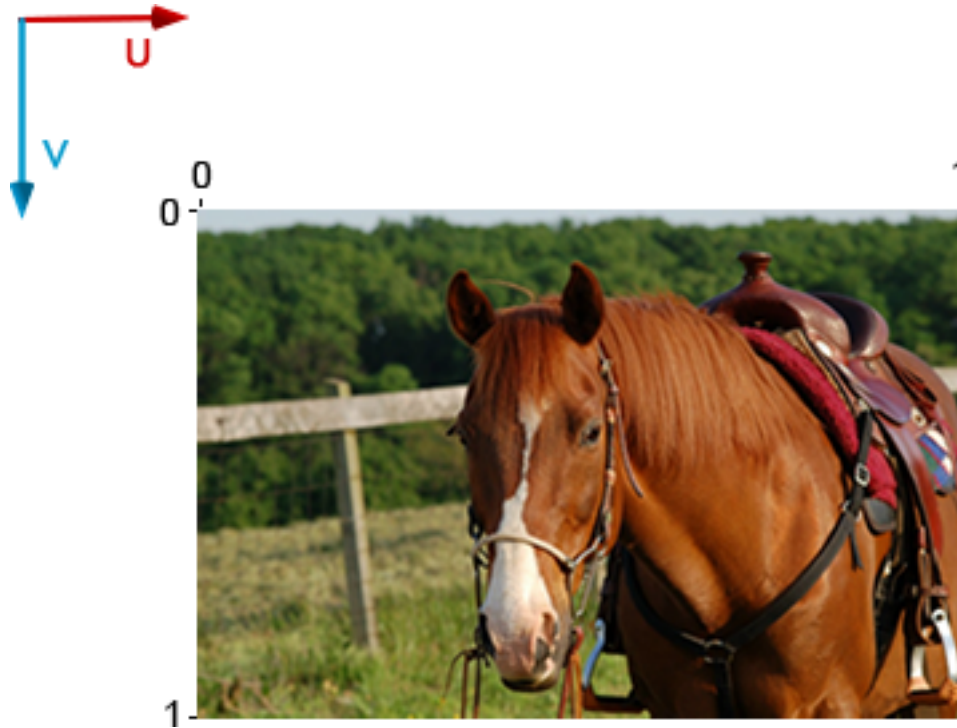
25,5,  
125,70,  
~~10,95,~~  
~~125,70,~~  
100,100,  
10,95

- **Indices:**

0,1,3,  
1,2,3

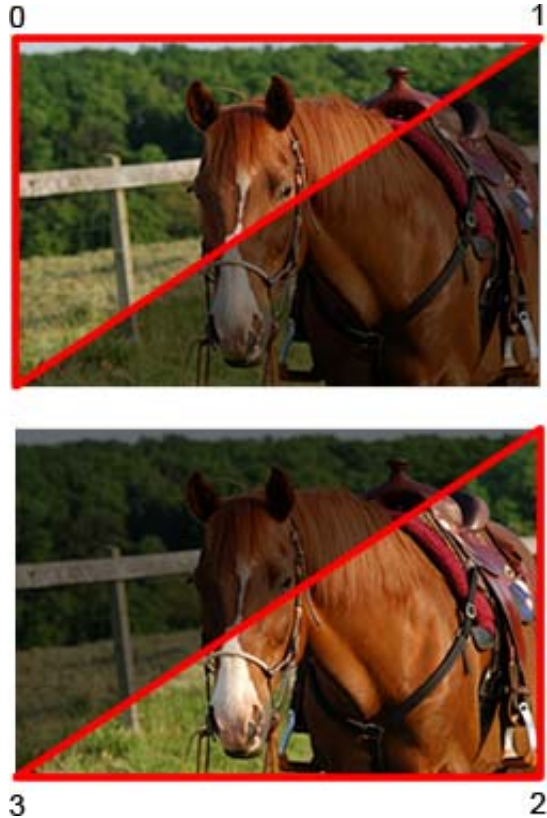


# 3D Without an Engine





# 3D Without an Engine



- **uvtData (UV):**

0,0,

1,0,

1,1,

0,1

(per vertex)



# 3D Without an Engine

`drawTriangles(vertices, indices, uvaData)`



# 3D Without an Engine

`drawTriangles(vertices, indices, uvaData)`





# 3D Without an Engine

UV = bitmap fit

T = perspective correction



# 3D Without an Engine

UV = bitmap fit

T = perspective correction

**UVT = bitmap fit with perspective**



# 3D Without an Engine

$$E = mc^2$$

$$T = \text{focal length} / (\text{focal length} + z)$$



# 3D Without an Engine

$$e = mc^2$$

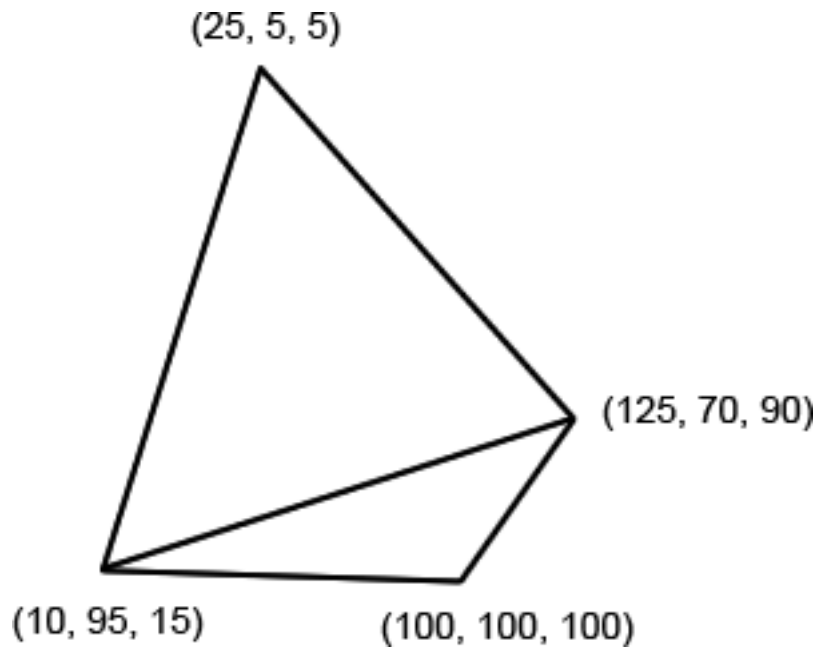
T = focal length / (focal length + z)

**T = proportion**





# 3D Without an Engine



- **uvtData (UVT):**

0,0, .9,

1,0, .3,

1,1, .25,

0,1, .8

(per vertex)



# 3D Without an Engine

`drawTriangles(vertices, indices, uvaData)`





# 3D Without an Engine

drawTriangles:

- Perspective rendering is **slower** than matrix transforms

but

- You can get cleaner results with far less geometry



# 3D Without an Engine

Hardly thinking:  
Flash Player 10 and  
**z, rotationX/Y/Z**



# 3D Without an Engine





# 3D Without an Engine

$$E = mc^2$$

rotationY += speed



# 3D Without an Engine

By Trevor McCauley  
(aka senocular)