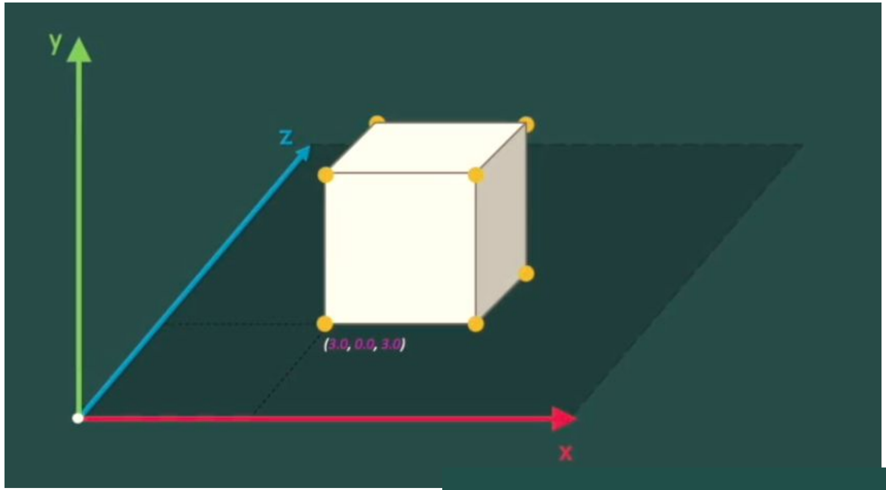
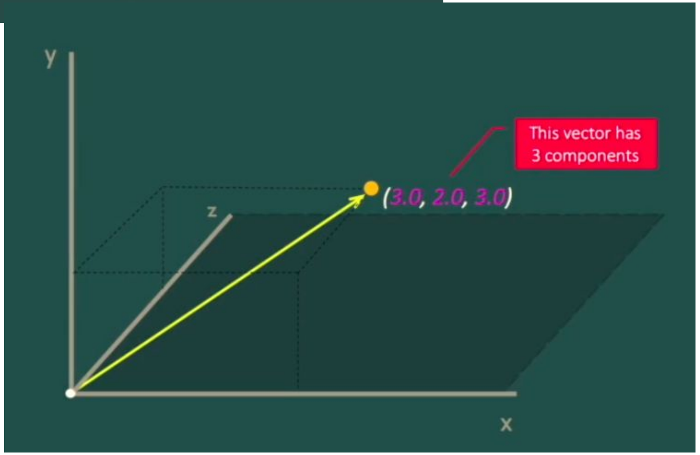
3D Graphics Programming from scratch in C w/ SDL

Part 1

Part 2: https://docs.google.com/presentation/d/1sOgXRzl_lRchMNssJKU7dyFZWkIS5JcUTDSdSF4gdD0/edit?usp=sharing

Rendering in a 3D space





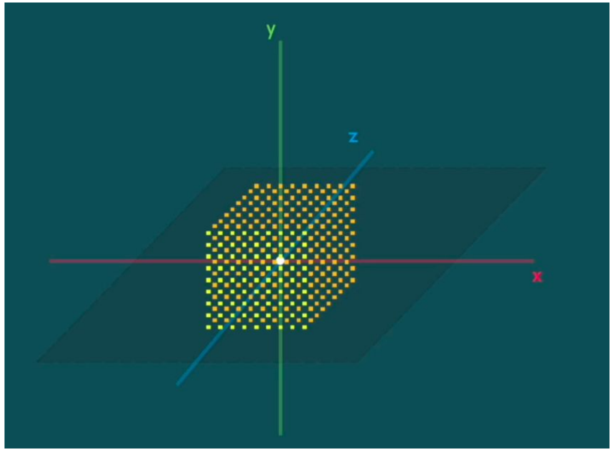
Since 3D space is *three* dimensional, 3D rendering all depends on vectors of size 3, meaning vectors with three elements. Each element represents a dimension e.g the first vertex on this cube equals:

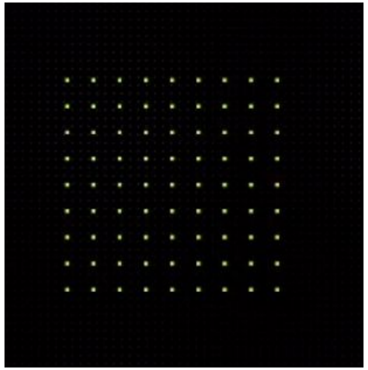
vec_1 = { x = 3, y = 0, z = 3 }.

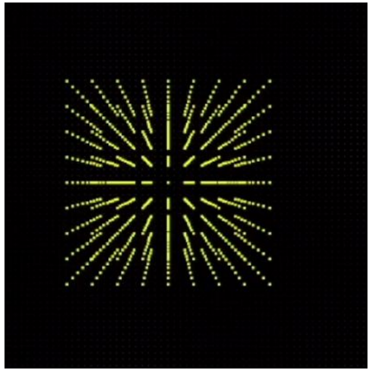
Think of the vector as starting at the origin (0, 0, 0) and ‘pointing’ to the vertex of the cube that it represents.

To render all the vertices that make up this cube as individual dots, you need an array of vectors, each with that dot’s co-ordinates.

Orthographic/Perspective projection







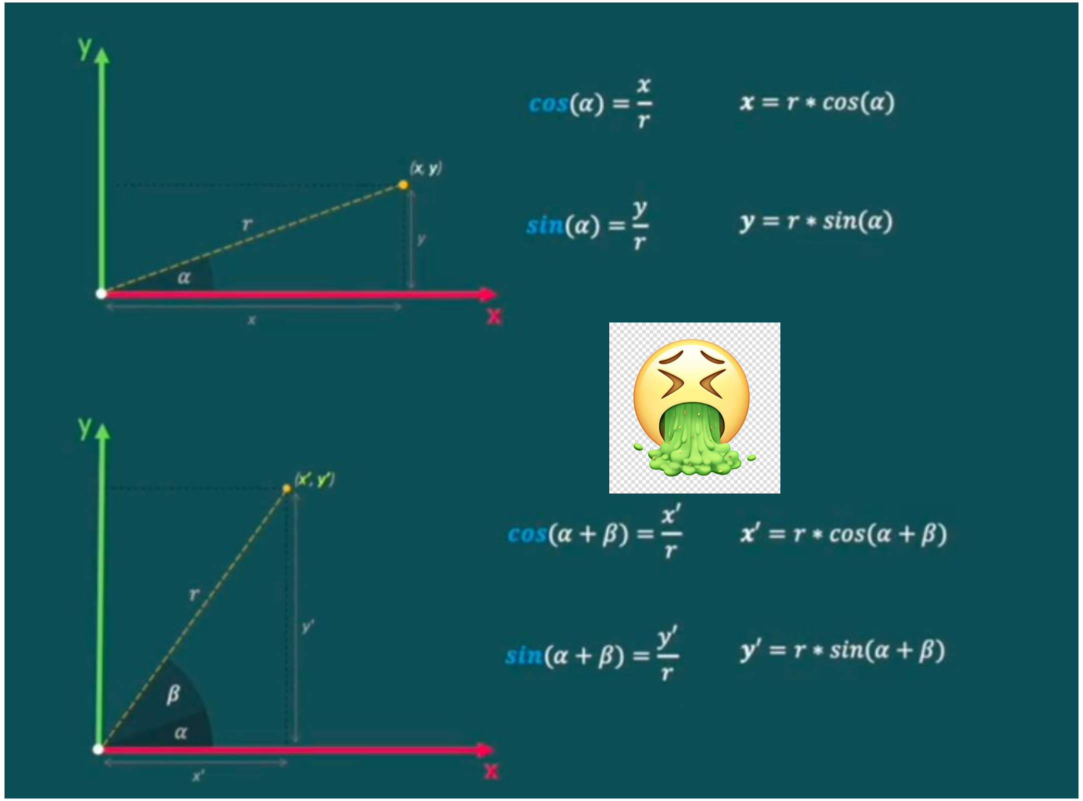
There are only 2 target render dimensions for the final image (since a screen is 2D), so once all the vector operations have taken place, the vectors need to be projected to 2D.

This is done by taking our final vector of length 3 and turning it into a vector of length 2, by dividing the X and Y by the Z. This scales the Xs and Ys in accordance with how far they should be from the screen.

Rotating & Repositioning vectors

A 3D object is no different to a picture if it doesn't move.

Moving (rotating) vectors involves a bunch of trig which I'm not going to subject you to, just trust me.

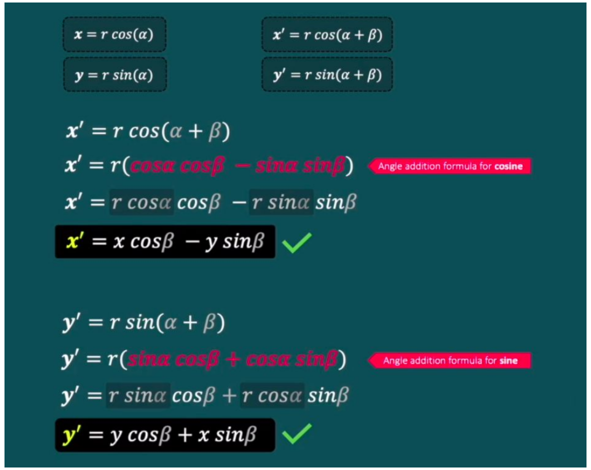


$\cos(\alpha) = \frac{x}{r}$ $\quad x = r * \cos(\alpha)$

$\sin(\alpha) = \frac{y}{r}$ $\quad y = r * \sin(\alpha)$

$\cos(\alpha+\beta) = \frac{x'}{r}$ $\quad x' = r * \cos(\alpha+\beta)$

$\sin(\alpha+\beta) = \frac{y'}{r}$ $\quad y' = r * \sin(\alpha+\beta)$



$x = r\cos(\alpha)$ $\quad x' = r\cos(\alpha+\beta)$

$y = r\sin(\alpha)$ $\quad y' = r\sin(\alpha+\beta)$

$x' = r\cos(\alpha+\beta)$

$x' = r(\cos\alpha\cos\beta - \sin\alpha\sin\beta)$ Angle addition formula for cosine

$x' = r\cos\alpha\cos\beta - r\sin\alpha\sin\beta$

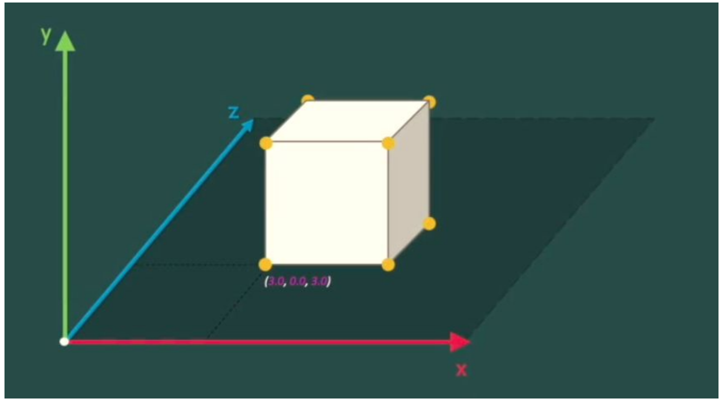
$\boxed{x' = x\cos\beta - y\sin\beta}$ ✓

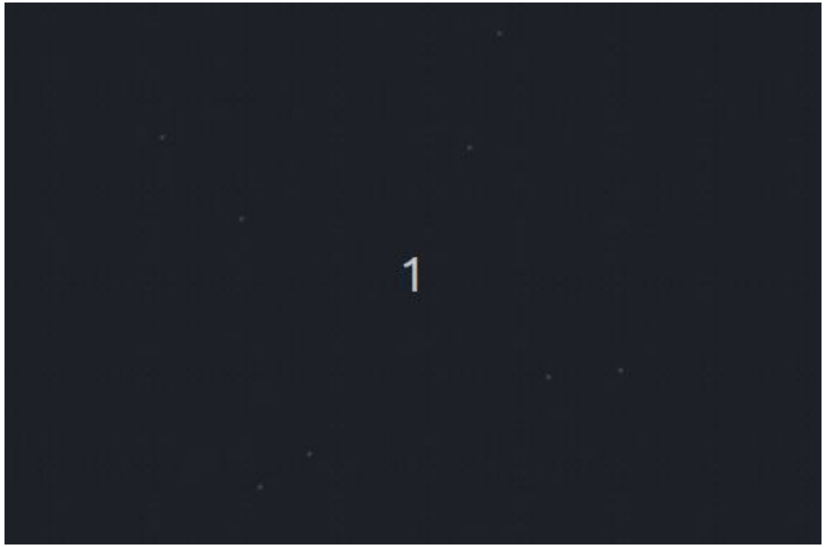
$y' = r\sin(\alpha+\beta)$

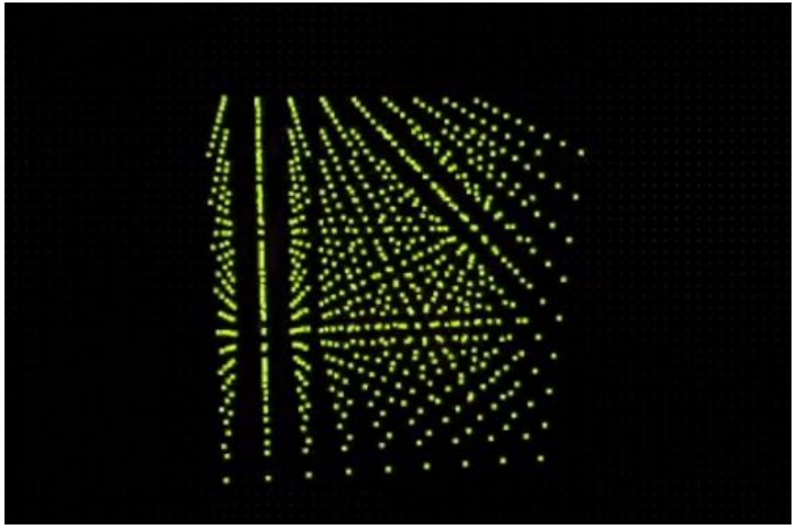
$y' = r(\sin\alpha\cos\beta + \cos\alpha\sin\beta)$ Angle addition formula for sine

$y' = r\sin\alpha\cos\beta + r\cos\alpha\sin\beta$

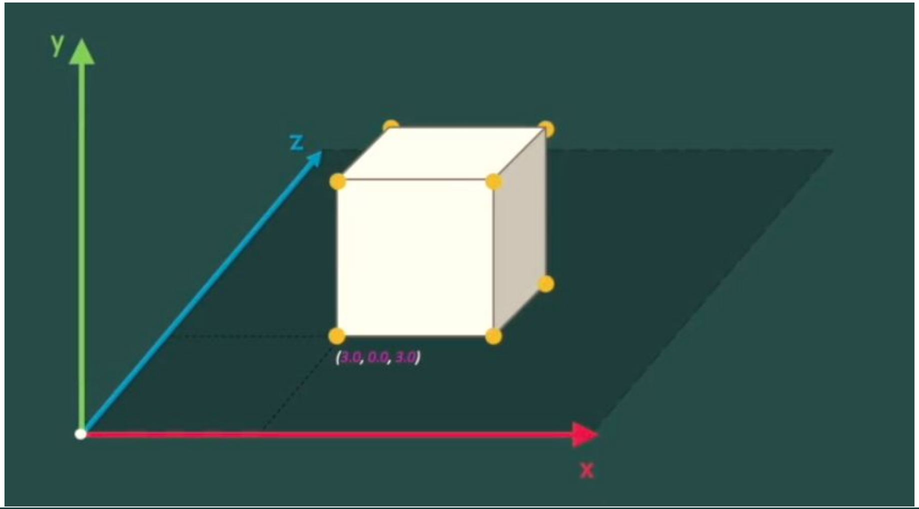
$\boxed{y' = y\cos\beta + x\sin\beta}$ ✓

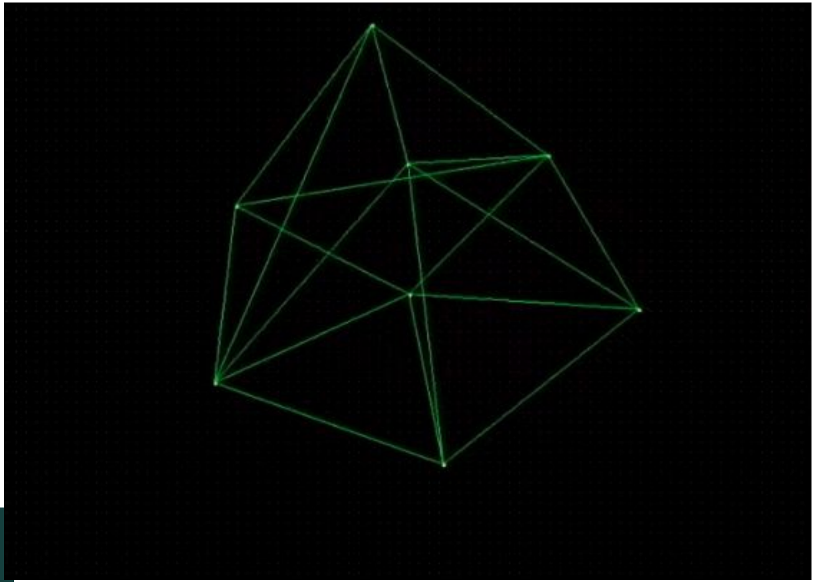

y
z
x
(3.0, 0.0, 3.0)

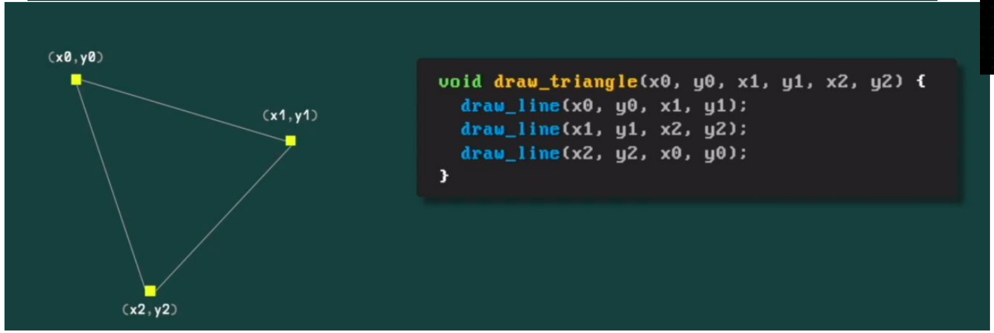

1



Triangles and Polygons

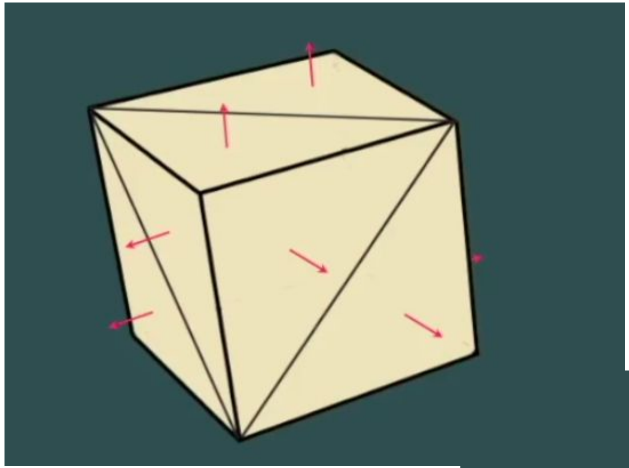


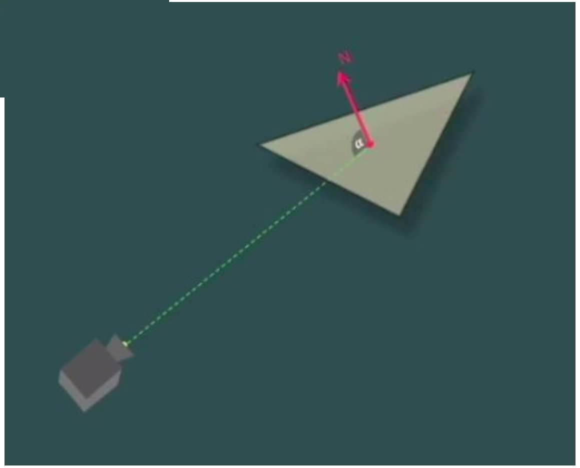




```
void draw_triangle(x0, y0, x1, y1, x2, y2) {
  draw_line(x0, y0, x1, y1);
  draw_line(x1, y1, x2, y2);
  draw_line(x2, y2, x0, y0);
}
```

Back-Face Culling

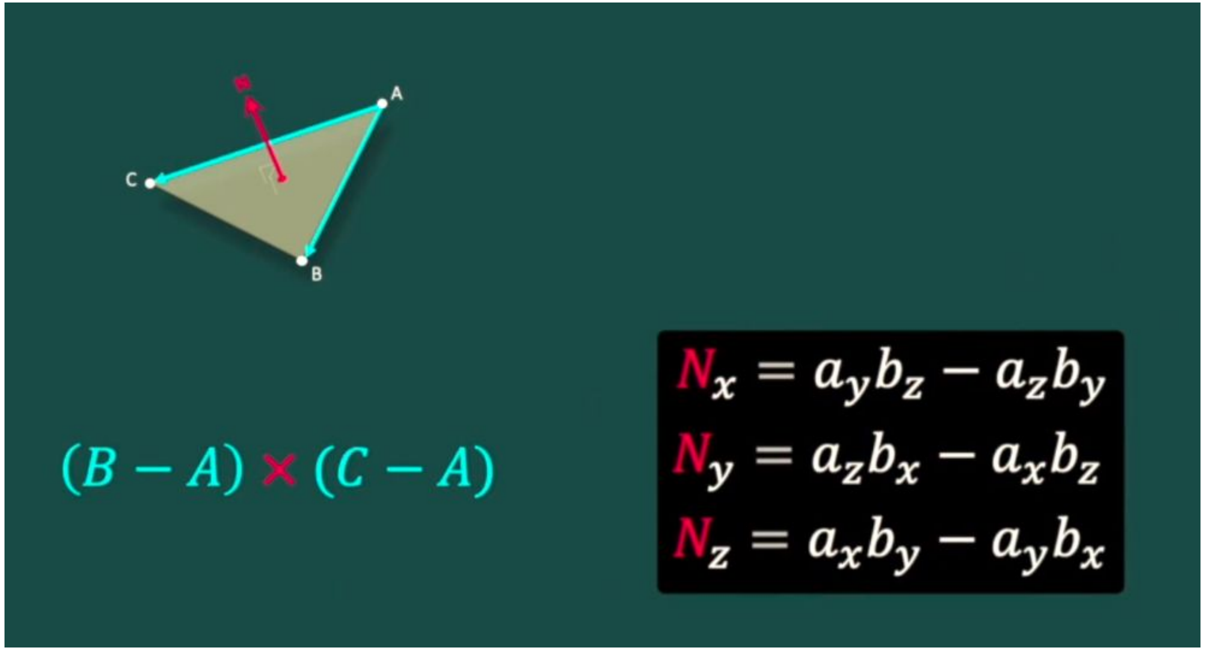




Back-face culling is the act of NOT rendering any triangles which are facing AWAY from the camera - that is, any triangles which make a greater angle between their face and the camera's view.

The vector projecting from a triangle's face is called the 'normal vector' of that polygon.

Normal vectors

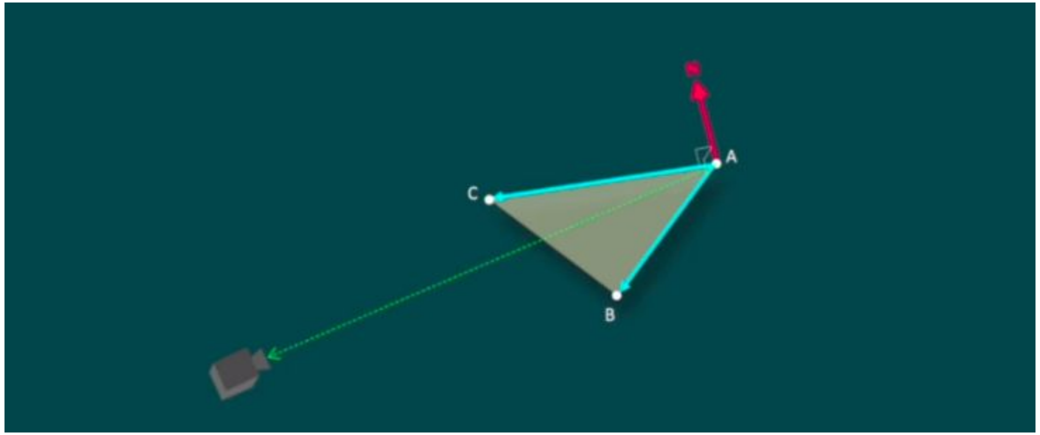


$$(B - A) \times (C - A)$$

$$N_x = a_y b_z - a_z b_y$$
$$N_y = a_z b_x - a_x b_z$$
$$N_z = a_x b_y - a_y b_x$$

The **normal vector** is the ‘cross-product’ of two vectors, which was the hardest part of the math to understand for me because I never did linear algebra at university. Could really have gone for a notation that is not the same as multiplication.

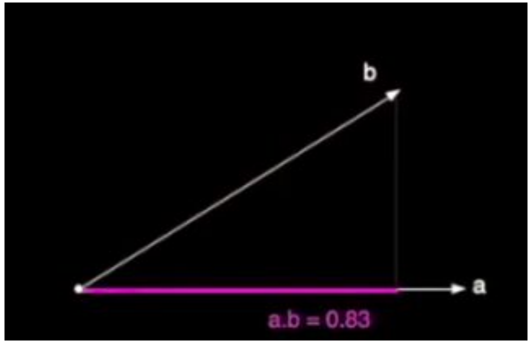
Dot product

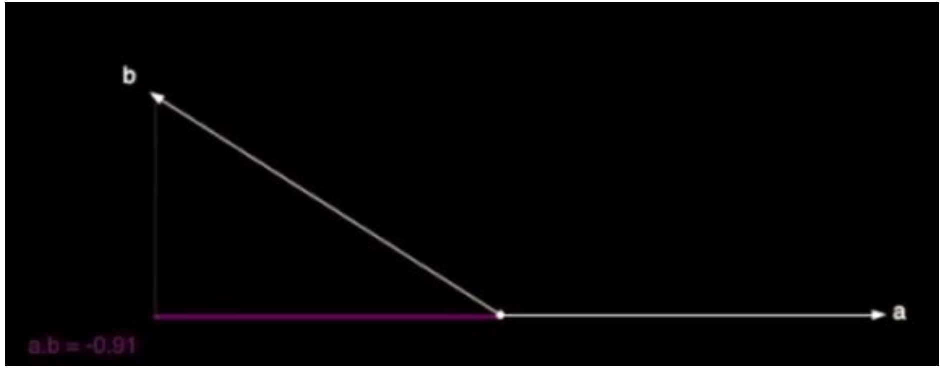


If the **dot product** (the ratio of directional similarity) of

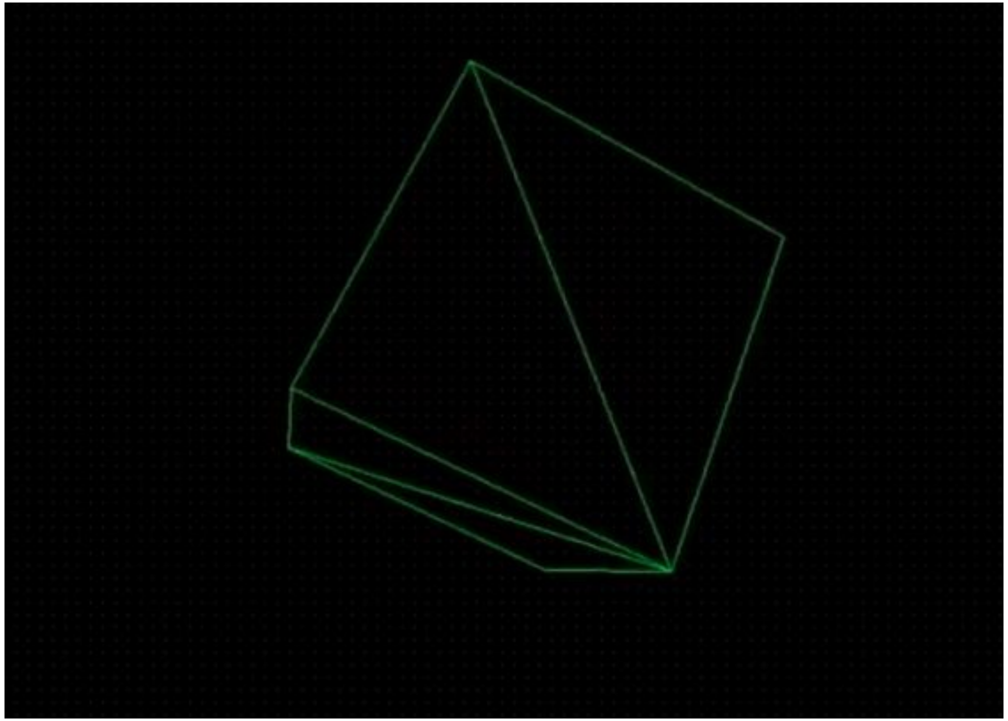
The camera ray vs. the normal vector

Is less than zero, don’t render that triangle.

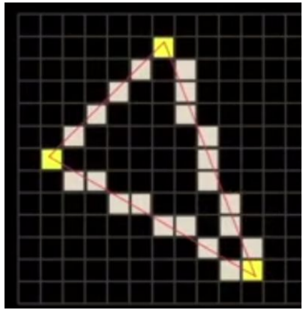


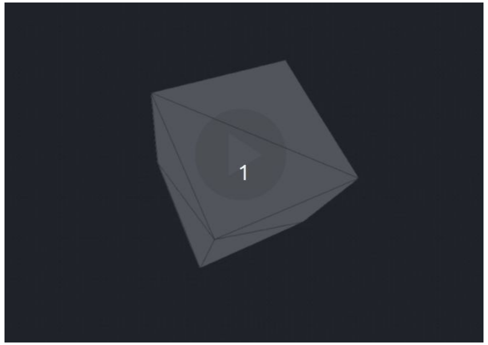


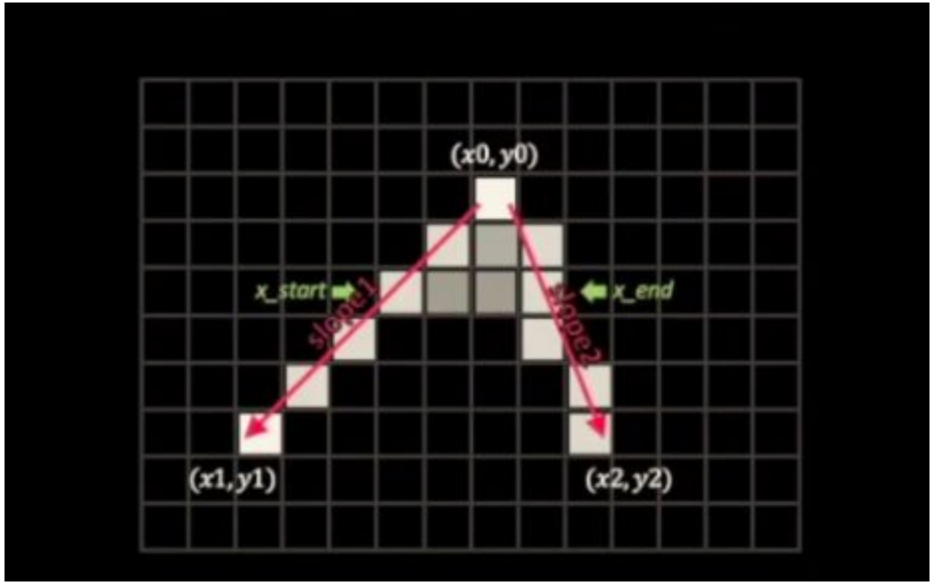
Hide all triangles not facing toward the camera



Filling the triangles







At this point, I’m only filling/texturing triangles with solid colors. The naive way this is done is by drawing individual lines between the pixels that form the edges of the triangles.

So, for every single row of pixels from the top of the triangle to the bottom, find the two pixels which have been drawn as part of the triangle, and draw a line between them.