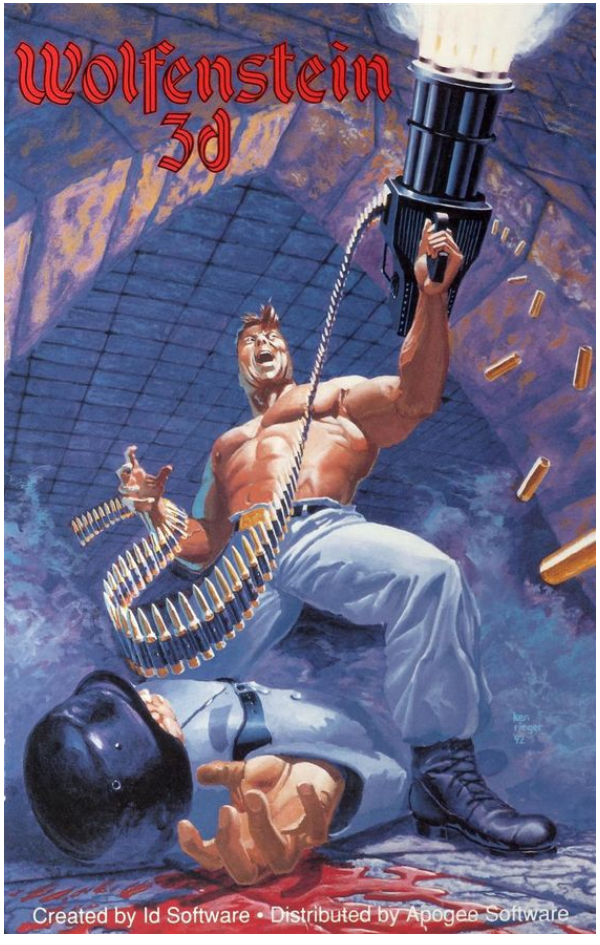
Caso de estudio en programación de videojuegos: Wolfenstein 3D (1992)

Miguel Angel Astor Romero

31 de octubre de 2017

Agenda

Introducción



Desarrollador id Software

Año 1992

Sistema MS-DOS

Wolfenstein 3D

Antecendentes: Maze War

Desarrollador Steve Colley, Greg Thompson y Howard Palmer

Año 1973

Sistema PDS-1

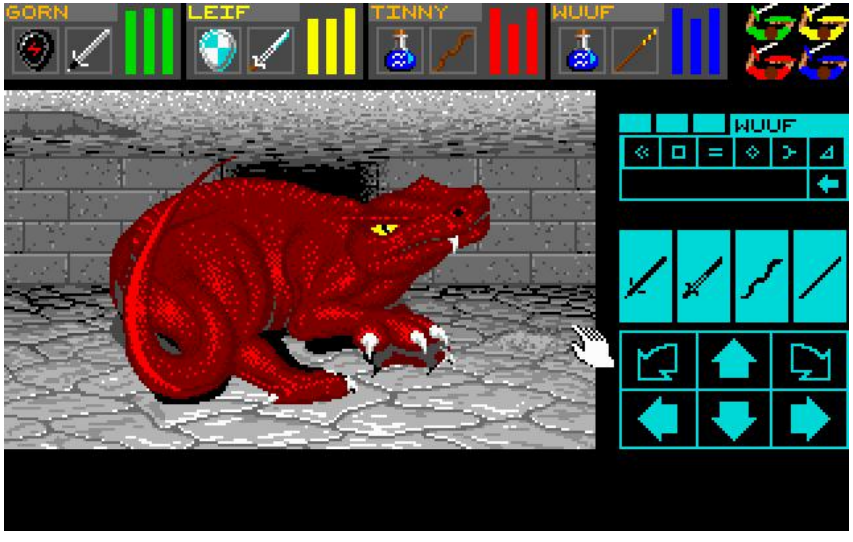
Antecedentes: 3D Monster Maze

Desarrollador J. K. Greye y Malcolm Evans

Año 1981

Sistema Sinclair ZX81

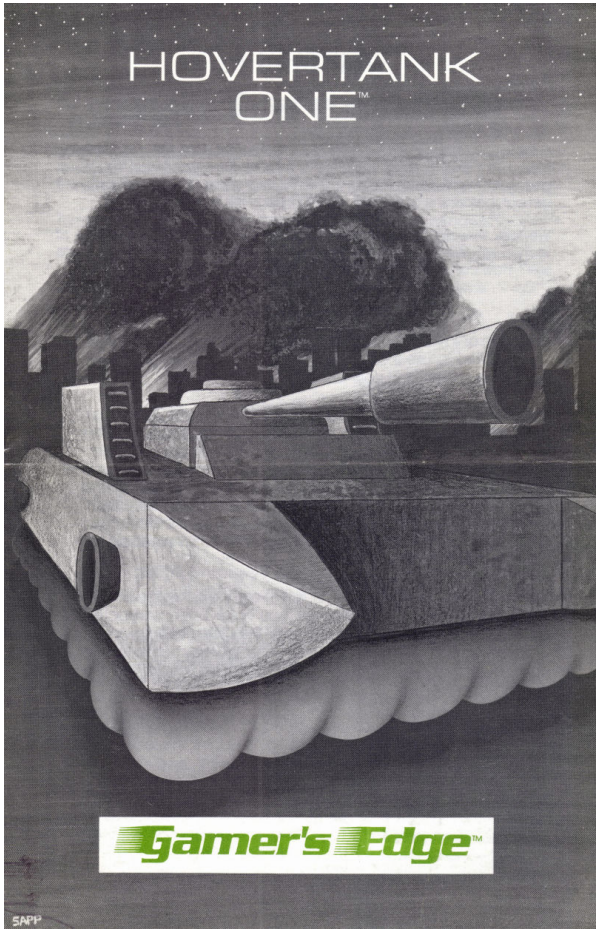
Otros antecedentes



Dungeon Master, FTL Games, Atari ST, 1987

Eye of the Beholder, Westwood, MS-DOS, 1991.

Hovertank One



Desarrollador id Software

Año 1991

Sistema MS-DOS

Ultima Underworld (1992)

Damas y caballeros: John Carmack!

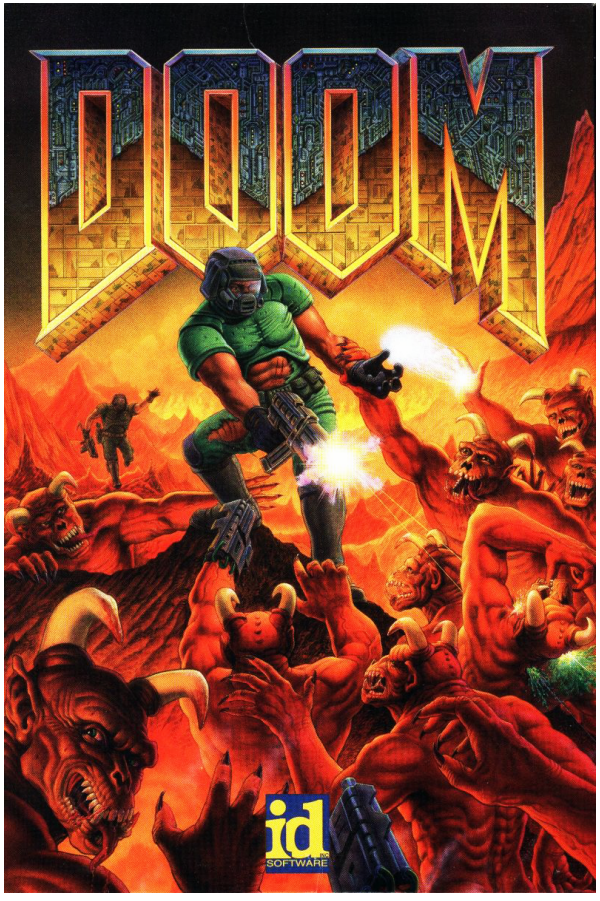
"Underworld shipped before Wolfenstein 3D. We had shown id a demo the year before, and I remember John Carmack (who was all of about 19 at the time, and as yet unknown in the games industry) saying that he could write a faster texture mapper..."

- Doug Church, Origin Systems, 2002.

El resultado de sus alardes: Catacombs 3D (1991)

Un año después

Descendientes: Doom (1993)

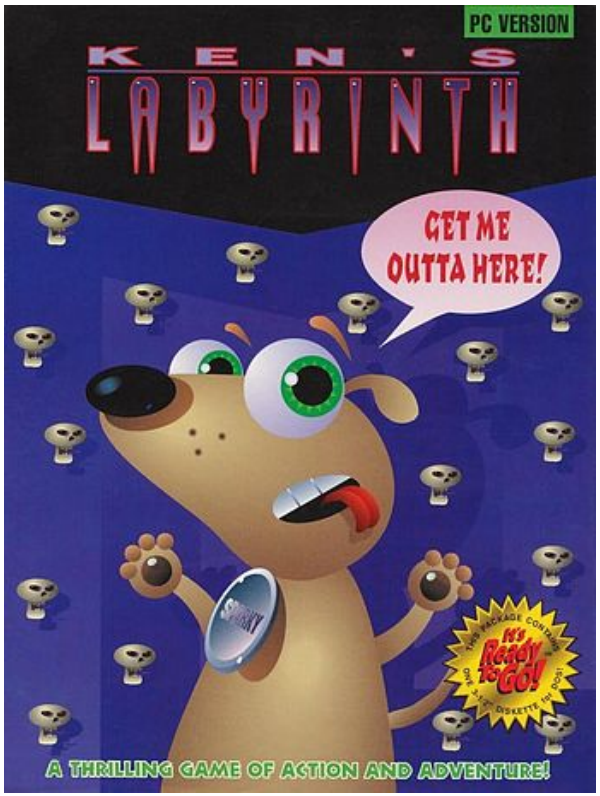


Desarrollador id Software

Año 1993

Sistema MS-DOS

Descendientes: Ken's Labyrinth



Desarrollador Ken Silverman

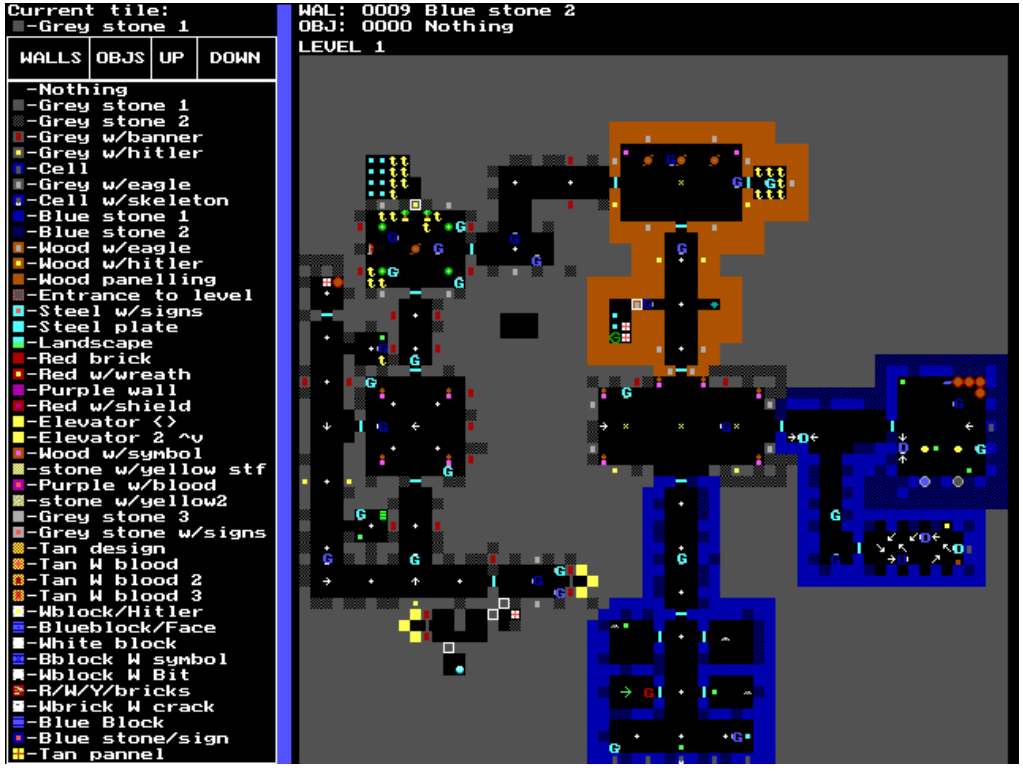
Año 1993

Sistema MS-DOS

Descendientes: Rise of the Triad

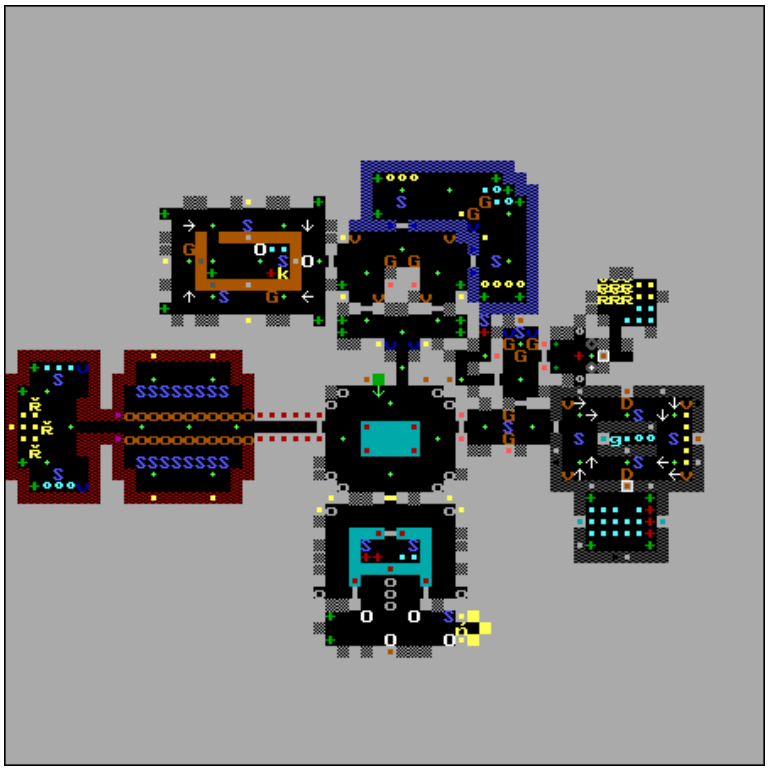
Apogee, MS-DOS, 1994

La mentira del 3D en Wolfenstein 3D



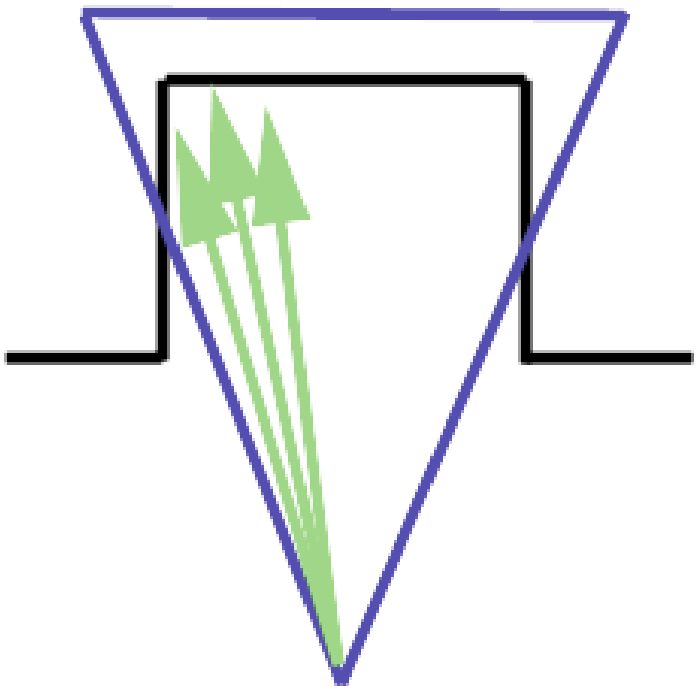
Entonces, ¿como desplegamos un mapa 2D en 3D?.

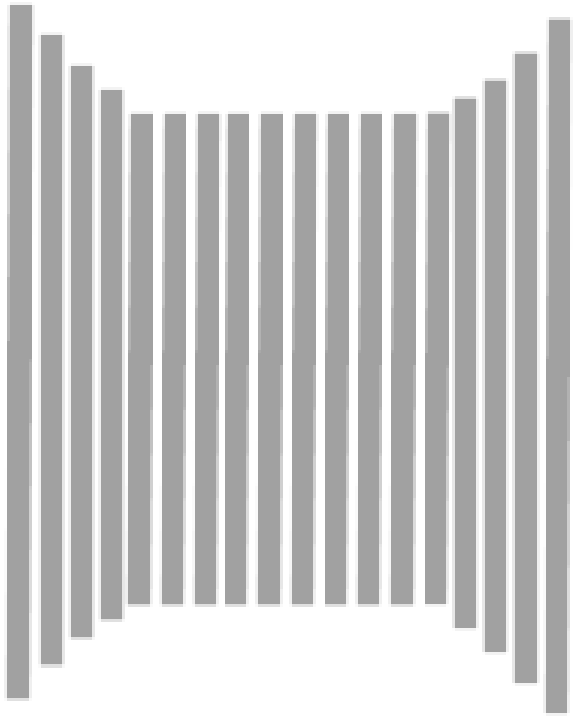
Se imponen restricciones al mundo



- El mundo se representa con una matriz.
- Las celdas pueden ser paredes o "vacías".
- Las celdas "vacías" pueden contener objetos o personajes.
- Las paredes siempre forman ángulos de 90 grados con el techo y el piso, y entre ellas.

Algoritmo de ray casting





Una simplificación muy drástica de Ray Tracing.

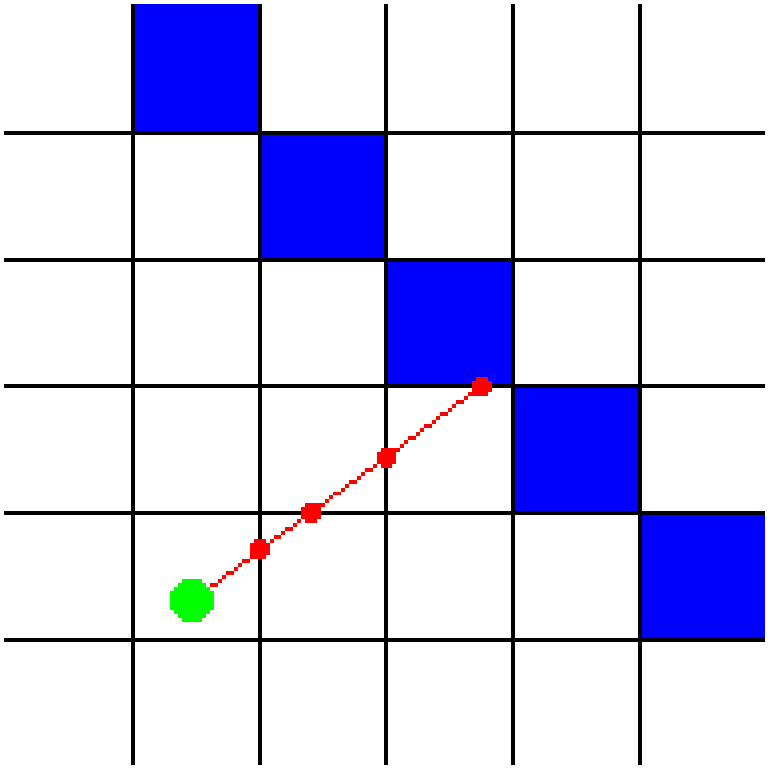
Rayos, rayos y más rayos

- Un rayo es un segmento de linea orientado con las siguientes propiedades:

$$r(t) = o + \vec{d}t; t \in [0, \infty)$$

- o Origen del rayo.
- $\vec{d}$ Dirección.
- t Desplazamiento.

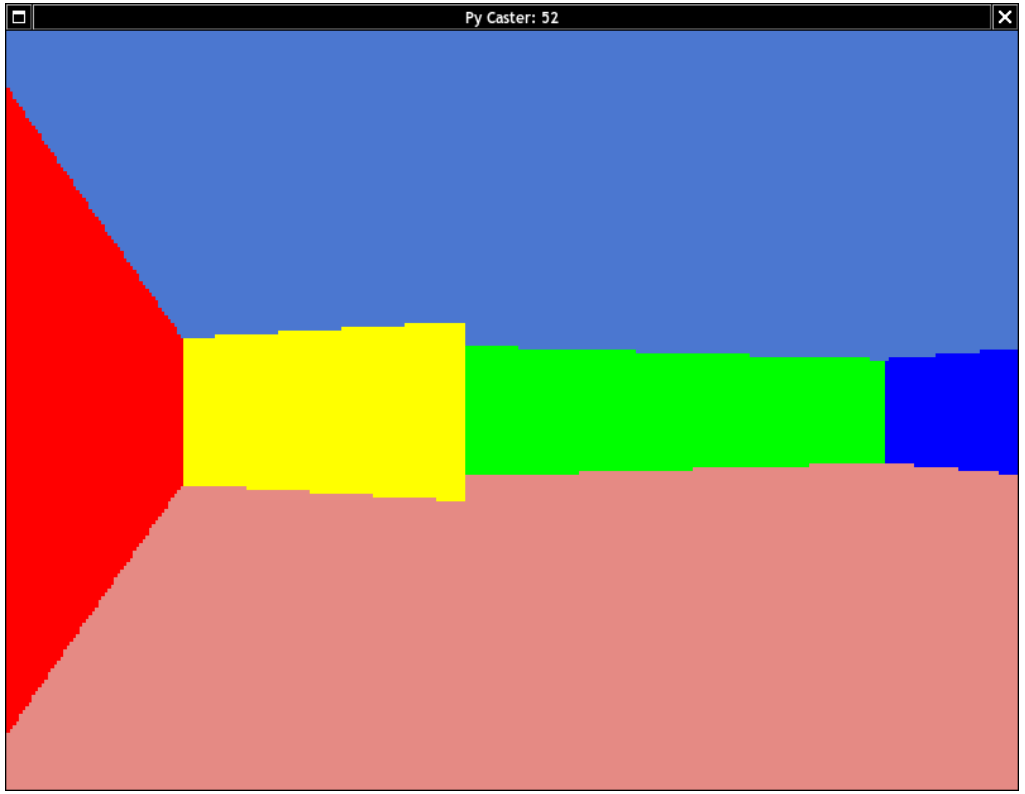
Intersecciones



La idea es encontrar las intersecciones del rayo con cada celda hasta que choque con una pared.

Generalización

Es posible eliminar la restricción de usar una matriz para representar el mundo, utilizando en su lugar segmentos de linea.

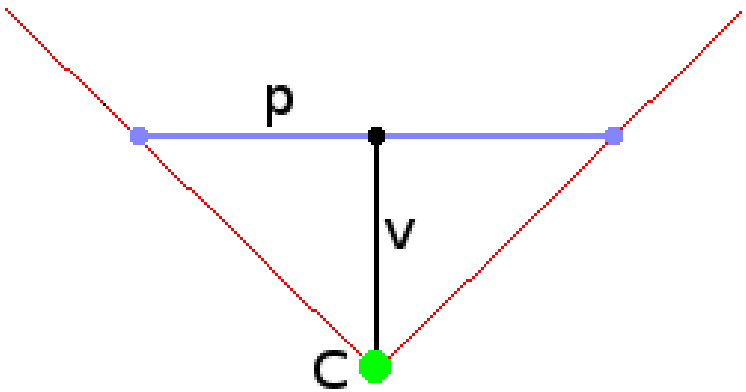


Algoritmo

En cada frame:

1. Por cada columna de pixeles de la pantalla
   1. Generar rayo
   2. Intersectar rayo con lineas
   3. Tomar intersección más cercana si existe
2. Si hay intersección
   1. Calcular altura de pared proyectada
   2. Dibujar pared

Generación de los rayos



Hay que definir tres elementos primero:

- La posición de la cámara c.
- La dirección de visión v.
- El vector de proyección p.

Generación de los rayos

Por cada columna de píxeles i se calcula t como sigue, donde W es el ancho de la pantalla en píxeles:

$$t = 2\frac{i}{W} - 1; t \in [-1, 1]$$

Luego el punto de origen del rayo esta dado por la posición de la cámara c, y su dirección estada dada por:

$$\vec{d} = (v_x + p_x t, v_y + p_y t)$$

Intersección rayo-segmento

Un segmento de recta entre dos puntos a y b se define como:

$$l(t) = a + (b - a)t; t \in [0, 1]$$

Si igualamos la ecuación del rayo con la del segmento:

$$r(t_1) = l(t_2) \Rightarrow o + t_1\vec{d} = a + (b - a)t_2$$

De aqui hay que despejar ecuaciones para t_1 y t_2:

$$t1 = \frac{|v_2 \times v_1|}{v_2 \cdot v_3}; t2 = \frac{v_1 \cdot v_3}{v_2 \cdot v_3}$$

Donde:

$$\vec{v1} = o - a; \vec{v2} = b - a; \vec{v3} = (-d_y, d_x)$$

Calculo de la altura de la pared proyectada

Conocidos $t1$ y el rayo $r(t) = o + \vec{d}t$, entonces el punto de intersección p viene dado por:

$$r(t_1) = o + \vec{d}t_1 = p$$

La distancia l entre el punto de intersección y la cámara es entonces:

$$l = \| p - o \|$$

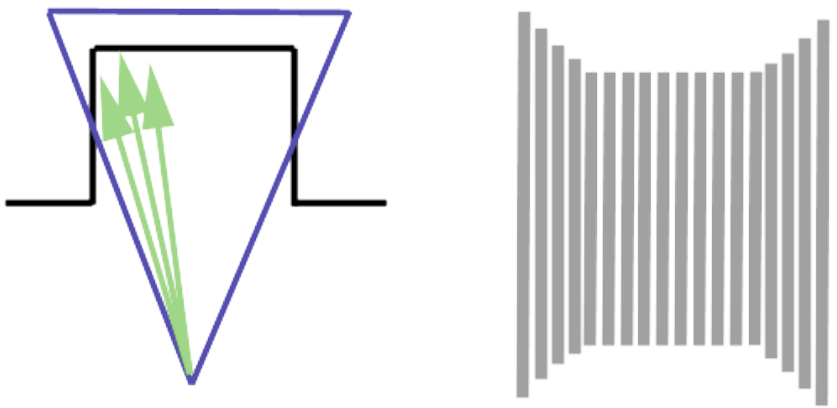
La altura en píxeles h de la pared en el plano de proyección está dada por la siguiente ecuación, donde H es la altura de la pantalla en píxeles:

$$h = \frac{H}{l}$$

Dibujado de la pared

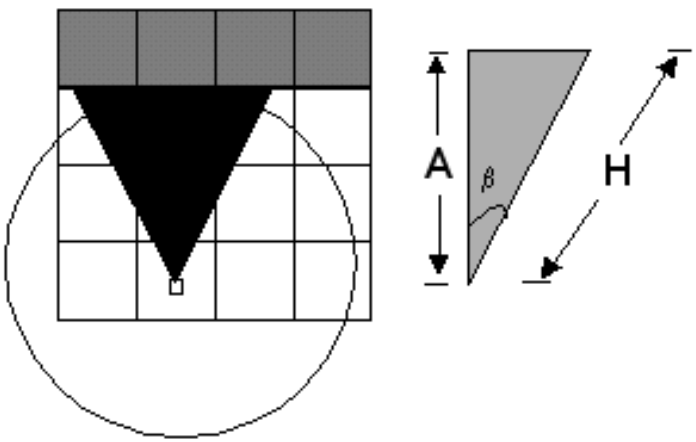
Asumiendo que la altura del jugador es exactamente la mitad de la altura de cualquier pared, entonces se tienen que colorear los pixeles de la columna correspondiente en el rango:

$$(-\frac{h}{2} + \frac{H}{2}, \frac{h}{2} + \frac{H}{2})$$



"Ojo de pez"

Corrección del efecto "ojo de pez"



Dado que:

$$\cos(\beta) = \frac{A}{H}$$

Entonces:

$$A = H\cos(\beta)$$

Posibles extensiones

- Aplicación de "texture mapping"
- "Floor casting" y "ceiling casting" para texturizar el suelo y techo, respectivamente.
- Suelos y techos de elevación variable.
- "Neblina".
- Sprites.
- Mirar hacia arriba o hacia abajo.

Referencias

- id Software, wolf3d, The original open source release of Wolfenstein 3D, github.com, 2012, consultado en octubre de 2017.
- P. Mallinson, Games that changed the world: Ultima Underworld, www.computerandvideogames.com, 2002, consultado en octubre de 2017.
- F. Sanglard, Let's Compile like it's 1992, fabiensanglard.net, 2014, consultado en octubre de 2017.
- F. Permadi, Ray-Casting Tutorial For Game Development And Other Purposes, permadi.com, 1996, consultado en octubre de 2017.
- L. Vandevenne, Lode's Computer Graphics Tutorial: Raycasting, lodev.org, 2007, consultado en octubre de 2017.

Contactos

Prof. Miguel A. Astor

- miguel.astor@ciens.ucv.ve
- miguel.a.astor@ucv.ve

¿Preguntas?