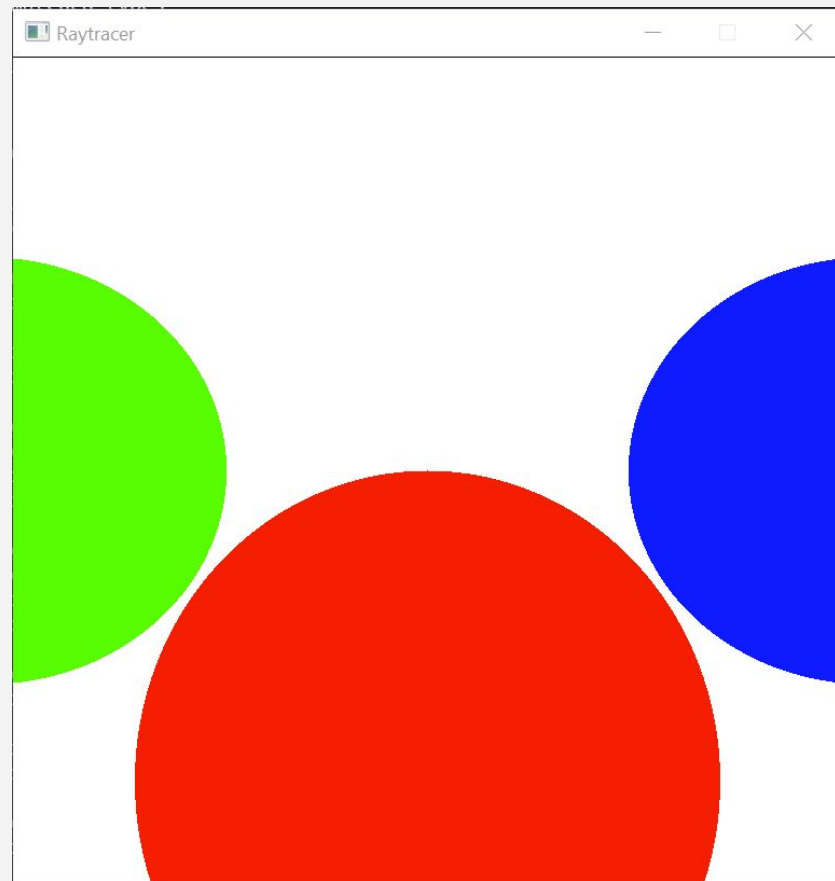


1 Minimal Raytracer

Based on *Computer Graphics from Scratch* by Gabriel Gambetta



High-level overview of a raytracing algorithm

Place the eye and the frame as desired ❶

For each square on the canvas

- ❷ Determine which square on the grid corresponds to this square on the canvas
- ❸ Determine the color seen through that grid square
- ❹ Paint the pixel with that color

- (1) Basic Assumptions
- (2) Canvas to Viewport Equation
- (3) Ray Equation + Ray and Sphere Intersection
- (4) canvas.PutPixel

Basic Assumptions

$$\text{camera position} = O = (\phi, \phi, \phi)$$

Forward is Z+, Up is Y+, Right is X+

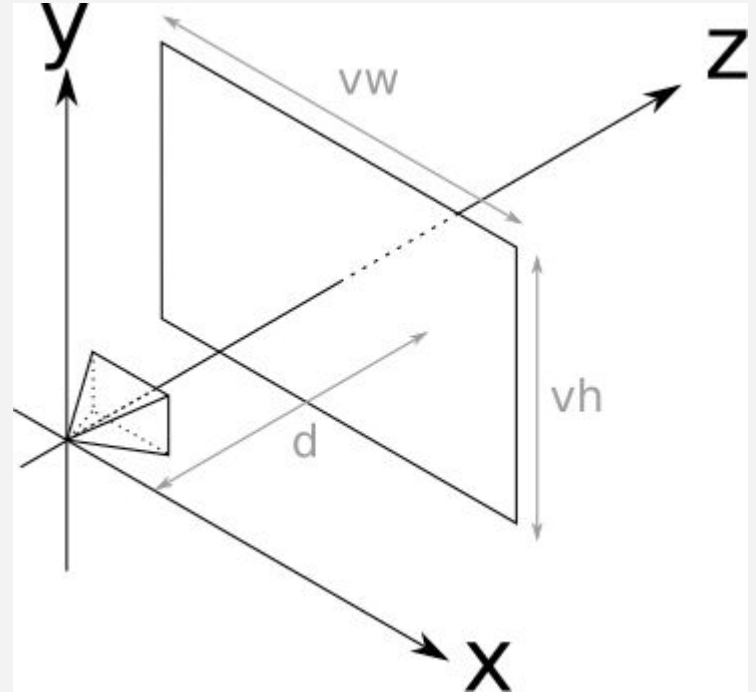
Viewport dimensions are **(Vw, Vh)**

Viewport is perpendicular to Z+

Viewport is at a distance **d**

Viewport sides are parallel to axes X and Y

$$V_w = V_h = d = 1$$



Canvas to Viewport Equation

$$V_x = C_x \cdot \frac{V_w}{C_w}$$

$$V_y = C_y \cdot \frac{V_h}{C_h}$$

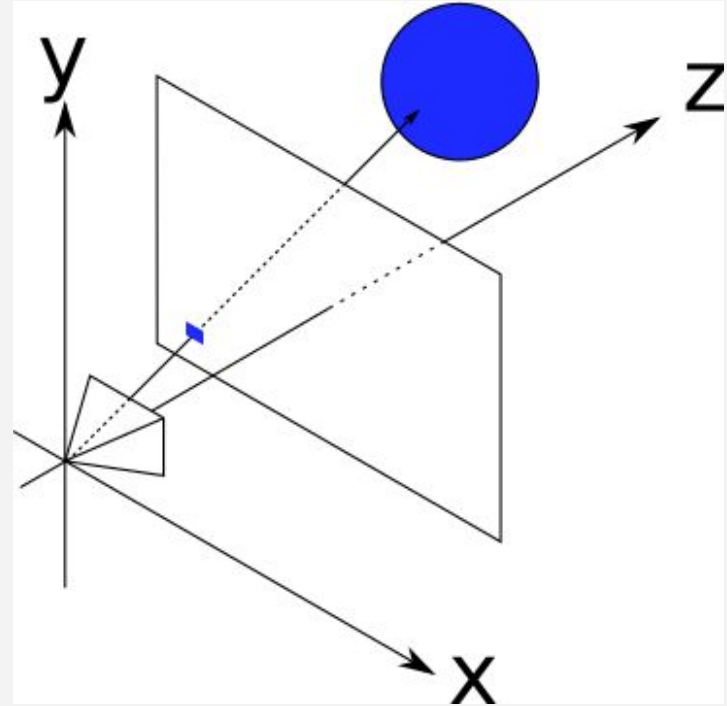
$$V_z = d$$

Going from canvas coordinates to viewport coordinates is just a **change of scale**, from pixels to world units.

The viewport is defined as being d units from the camera, so **V_z is always d** .

Determining the color for each pixel

In order to know which color to paint each pixel, we send rays starting from the camera origin and crossing the viewport plane. We then record which objects the ray intersects and obtain the color from the closest one.



The Ray Equation

$$P = O + t(V - O)$$

$$P = O + t\vec{D}$$

P is each point of the ray, and t is a continuous variable.

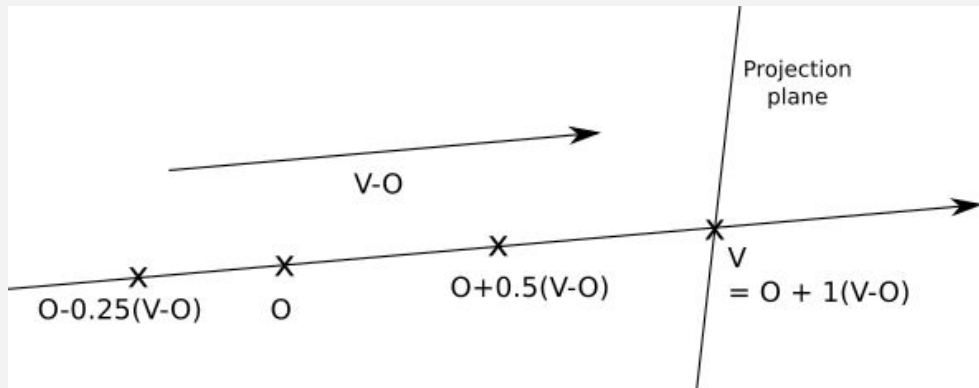
If we call $(V - O)$, the direction of the ray, D , we get the second equation

The equation starts at O and advances in the direction D .

When $t < 0$, P is behind the camera

When $0 < t < 1$, P is between the camera and the projection plane / viewport

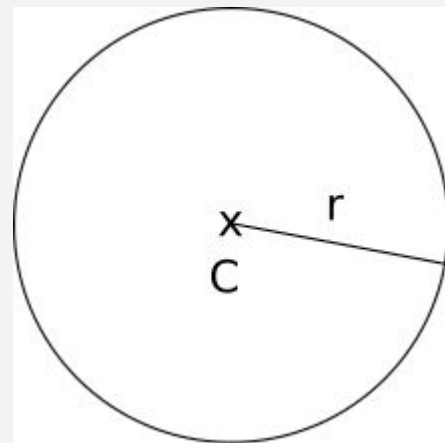
When $t > 1$, P is in front of the viewport



The Sphere Equation

A sphere is defined by its center, C , and a radius, r . Every point P in the sphere's surface satisfies that the distance between itself and the center equals r .

$$|P - C| = r$$



The length of a vector is the square root of its dot product with itself

$$\sqrt{\langle P - C, P - C \rangle} = r$$

$$\langle P - C, P - C \rangle = r^2$$

Ray meets Sphere

We are just solving a system of equations, where t and P are the variables.

$$\langle P - C, P - C \rangle = r^2$$
$$P = O + t\vec{D}$$

$$\langle O + t\vec{D} - C, O + t\vec{D} - C \rangle = r^2$$

$$\langle \vec{CO} + t\vec{D}, \vec{CO} + t\vec{D} \rangle = r^2$$

$$\langle \vec{CO} + t\vec{D}, \vec{CO} \rangle + \langle \vec{CO} + t\vec{D}, t\vec{D} \rangle = r^2$$

$$\langle t\vec{D}, t\vec{D} \rangle + 2\langle \vec{CO}, t\vec{D} \rangle + \langle \vec{CO}, \vec{CO} \rangle = r^2$$

$$t^2 \langle \vec{D}, \vec{D} \rangle + t(2\langle \vec{CO}, \vec{D} \rangle) + \langle \vec{CO}, \vec{CO} \rangle - r^2 = 0$$

$$at^2 + bt + c = r^2$$

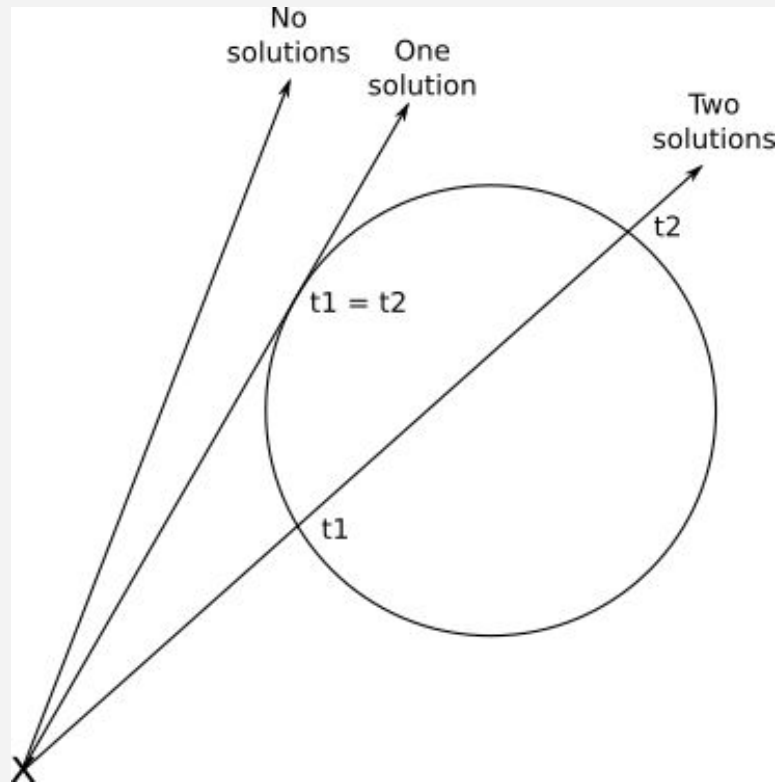
Ray meets Sphere 2

We end up with a quadratic equation that gives us two solutions for t .

This makes geometric sense, because the ray can intersect a sphere on two points.

We can substitute the solution for t to get the value of P .




We only project into the viewport the objects that are found for values of $t > 1$, all others are considered to be behind the camera.



Source code

A C implementation of the algorithm is at:

<https://github.com/tato/Raytracer/blob/eecf088fc12f7dc4b4a458b3c9a9487b172faa86/ray/ray.c>

```
124 lines (98 sloc) | 2.97 KB
Raw Blame   

1  #include <ray.h>
2
3  #include <stdbool.h>
4  #include <math.h>
5
6  #define INF 1000000.f
7
8  typedef struct {
9      float x, y, z;
10 } V3;
11
12 typedef struct {
13     V3 center;
```