# Vizualizing math

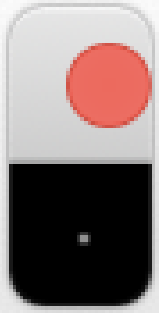
Using math for pictures, using pictures to explain the math.

July 20, 2022

# Drawing on your computer

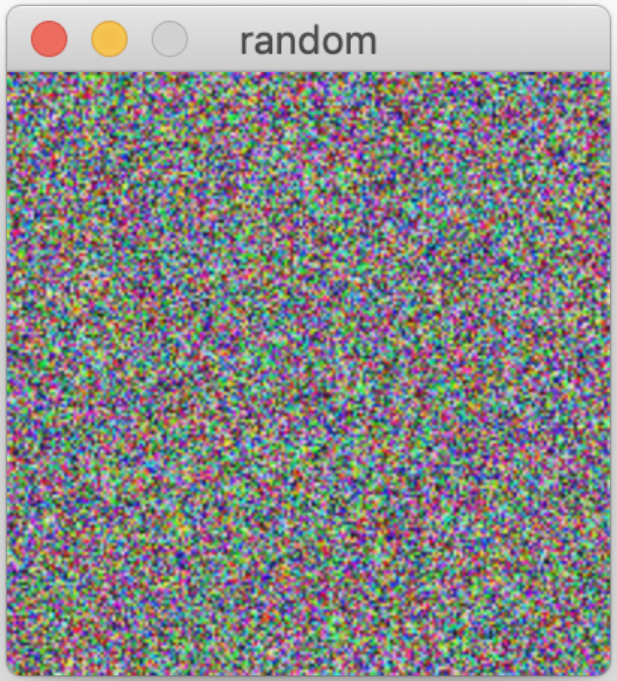
# The color of a pixel

```
····pixel_put_image(&(info.img), 10, 10, 0x00ffffff);
```

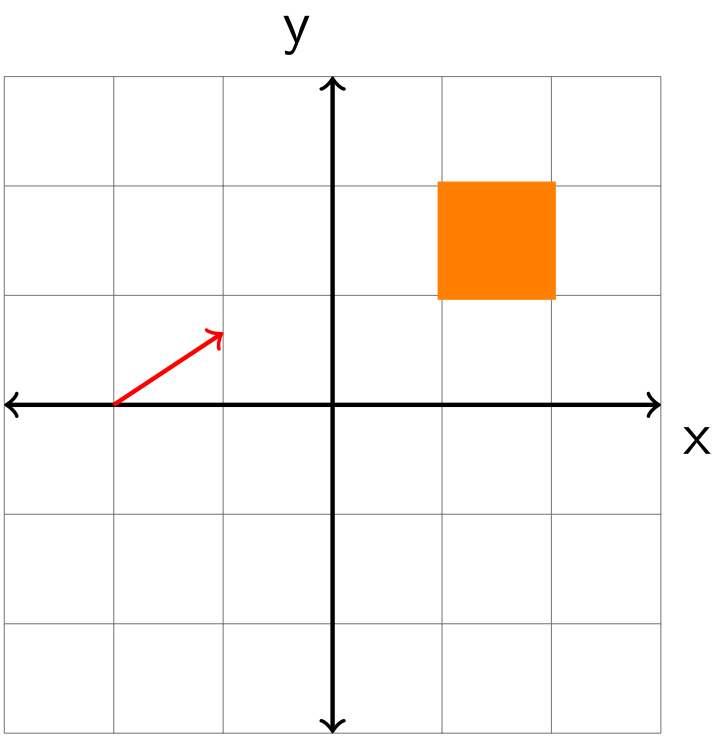


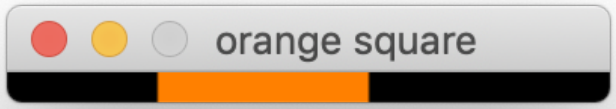
# Let the computer do graphics

```
    for (int i = 0; i < 200; i++)
        for (int j = 0; j < 200; j++)
            pixel_put_image(&(info.img), i, j, rand());
```



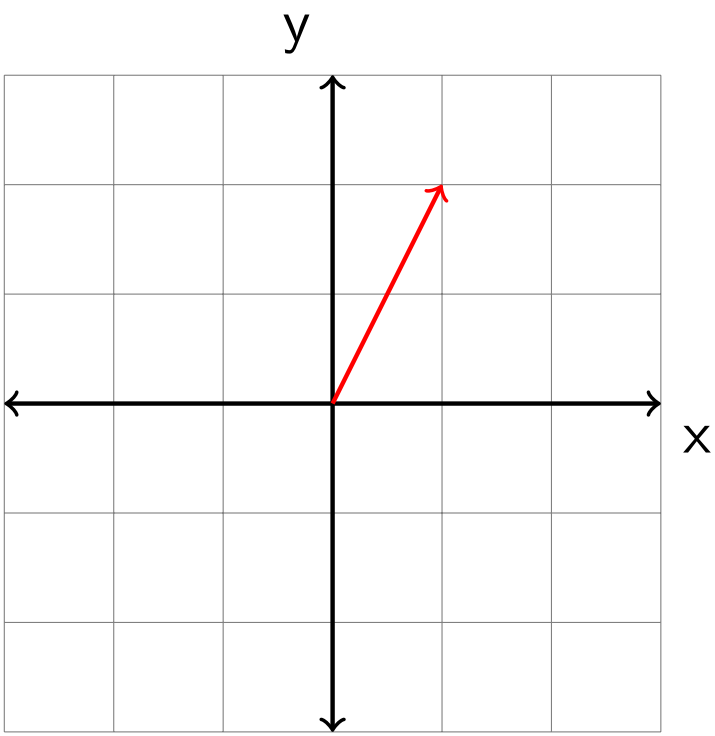
# What am I looking at



# Vector, representation

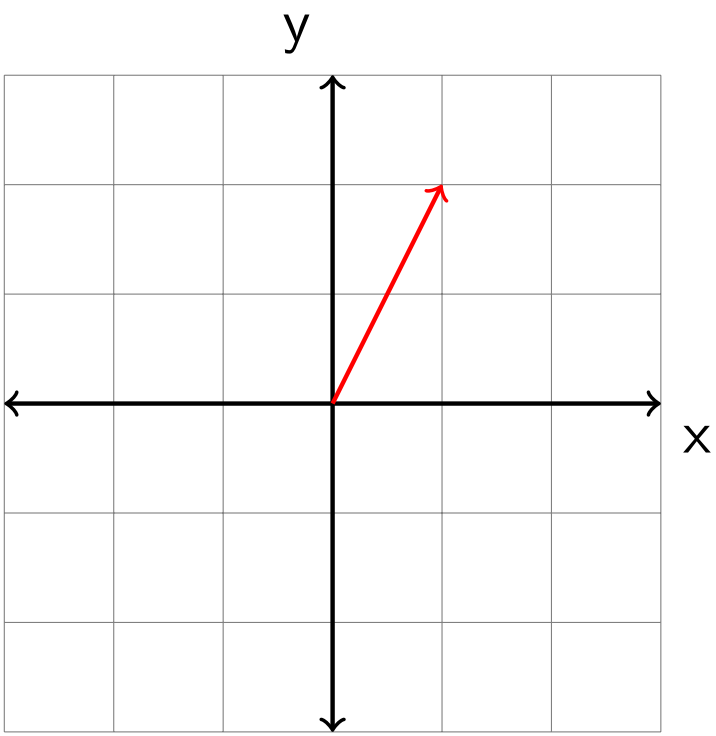
$$\vec{V} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$



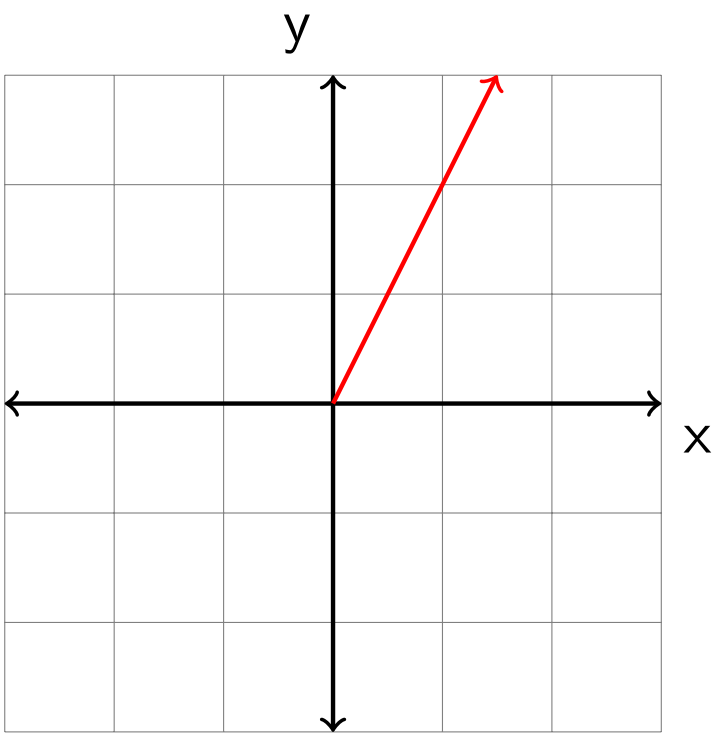

```
struct vec {
    float x;
    float y;
};

void somefunction(struct vec v) {
    v.x = 1;
    v.y = 2;
}
```

# Scaling a vector

$$\vec{V} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$




$$1.5 * \vec{V} = \begin{bmatrix} 1.5 \\ 3 \end{bmatrix}$$

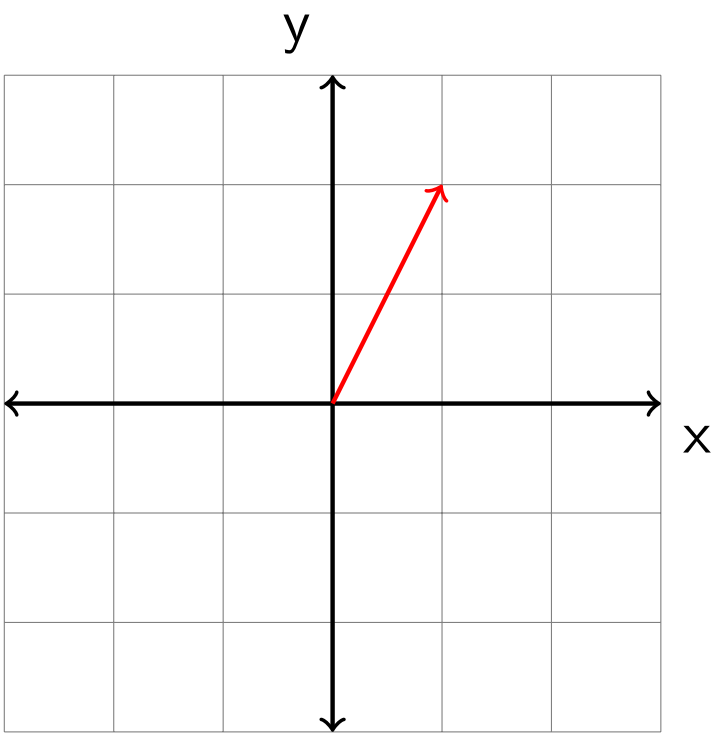


# Code for scaling a vector

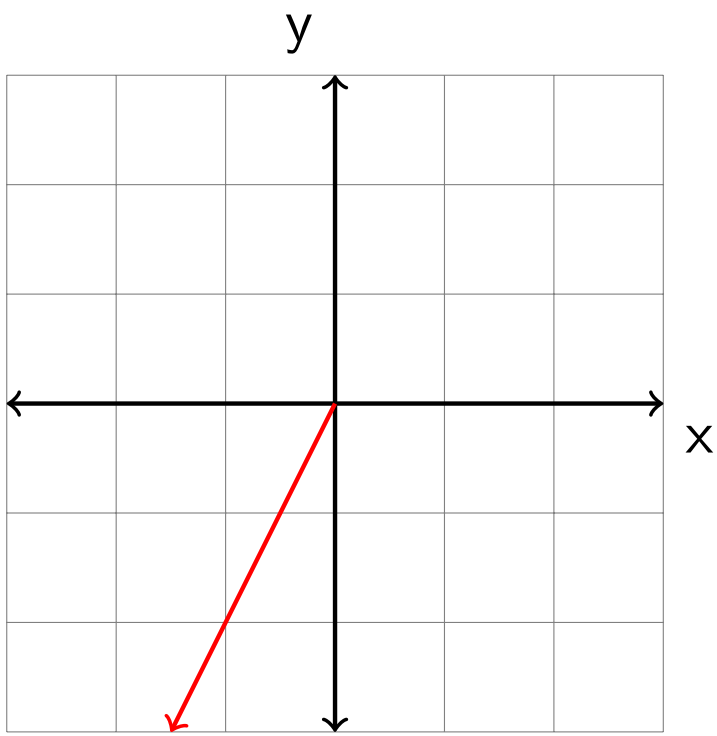
```
struct vec scale(struct vec v, float scale) {
    v.x *= scale;
    v.y *= scale;
    return v;
}
```

# Scaling with negative number

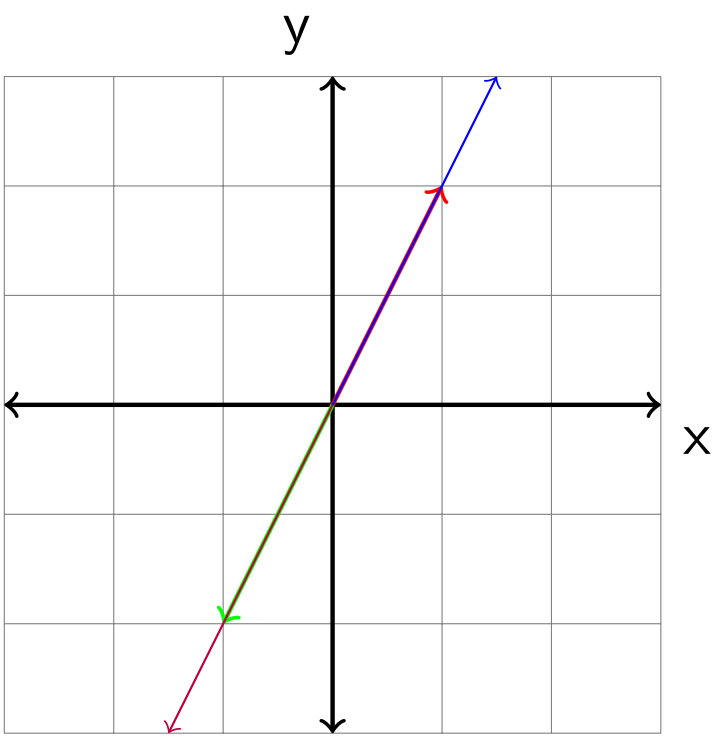
$$\vec{V} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$




$$-1.5 * \vec{V} = \begin{bmatrix} -1.5 \\ -3 \end{bmatrix}$$



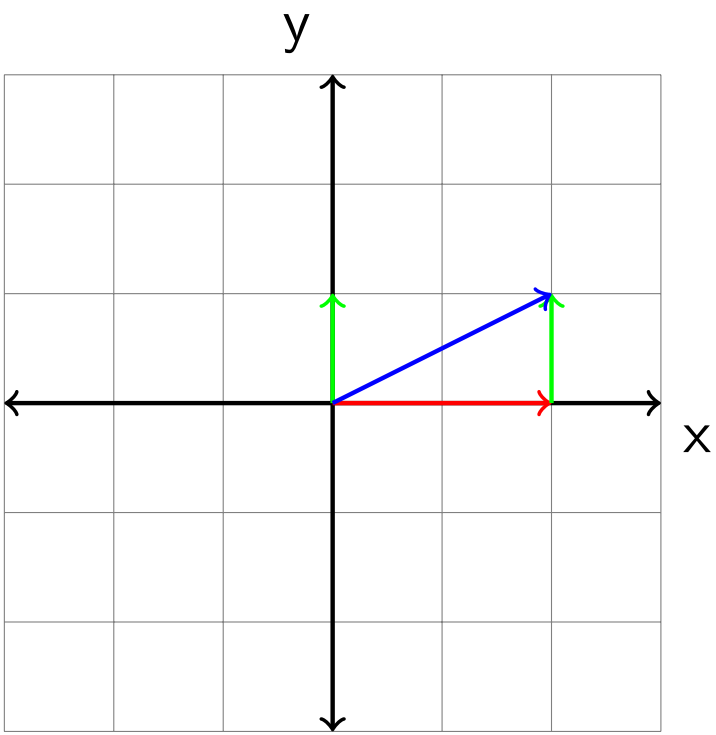
# Scaling all happens on the same line




$t * \vec{v}$

# Adding vectors, or splitting up in components

$$\begin{bmatrix} 2 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

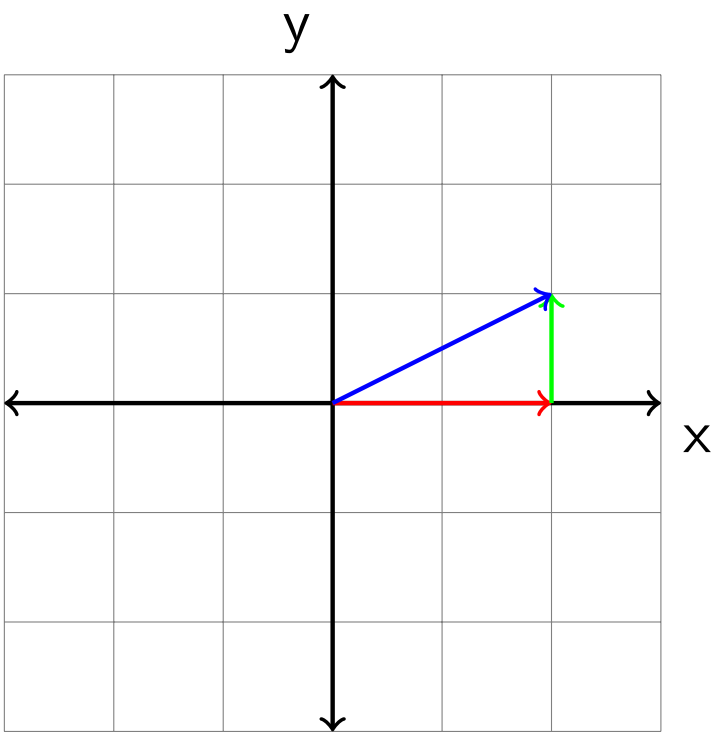


# Code for adding vectors

```
struct vec add(struct vec v1, struct vec v2) {
    v1.x += v2.x;
    v1.y += v2.y;
    return v1;
}
```

# Subtracting vectors

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} - \begin{bmatrix} 2 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

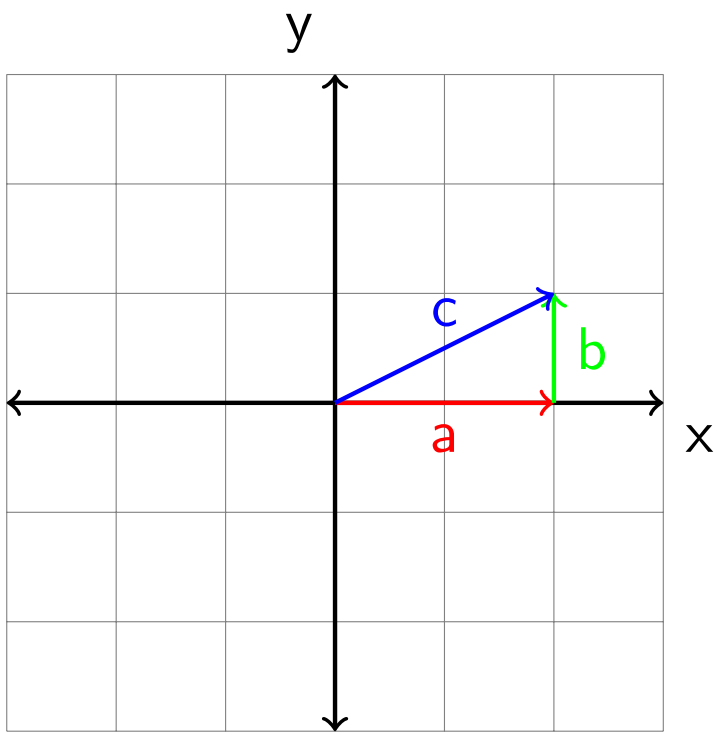


# Inner product

$$\begin{bmatrix} 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix} = 2 * 3 + 1 * 4$$

$$\vec{c}^2 = \begin{bmatrix} a \\ b \end{bmatrix} \cdot \begin{bmatrix} a \\ b \end{bmatrix} = a * a + b * b$$

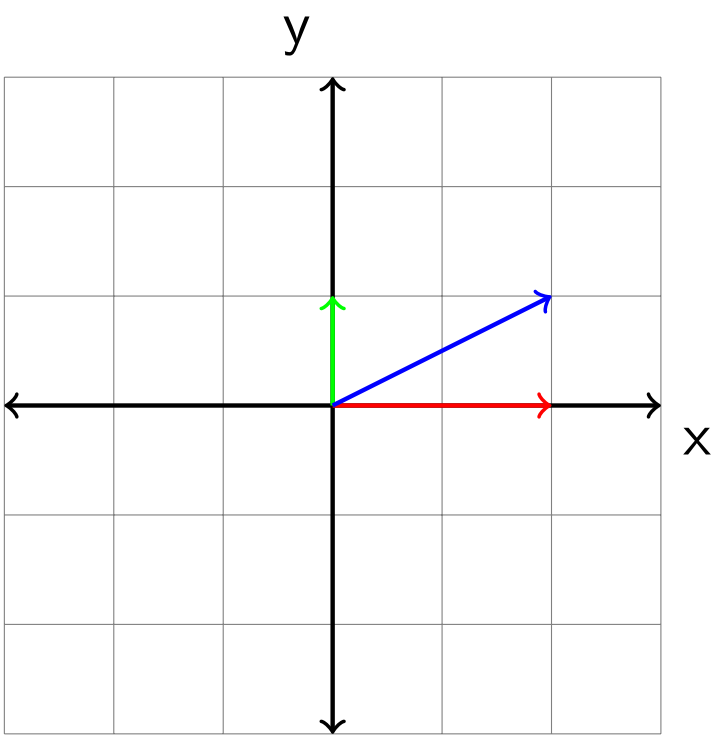
$$a^2 + b^2 = c^2$$



# Inner product, projection

$$\begin{bmatrix}1\\0\end{bmatrix} \cdot \begin{bmatrix}2\\1\end{bmatrix} = 1 * 2 + 0 * 1 = 2$$

$$\begin{bmatrix}0\\1\end{bmatrix} \cdot \begin{bmatrix}2\\1\end{bmatrix} = 0 * 2 + 1 * 1 = 1$$

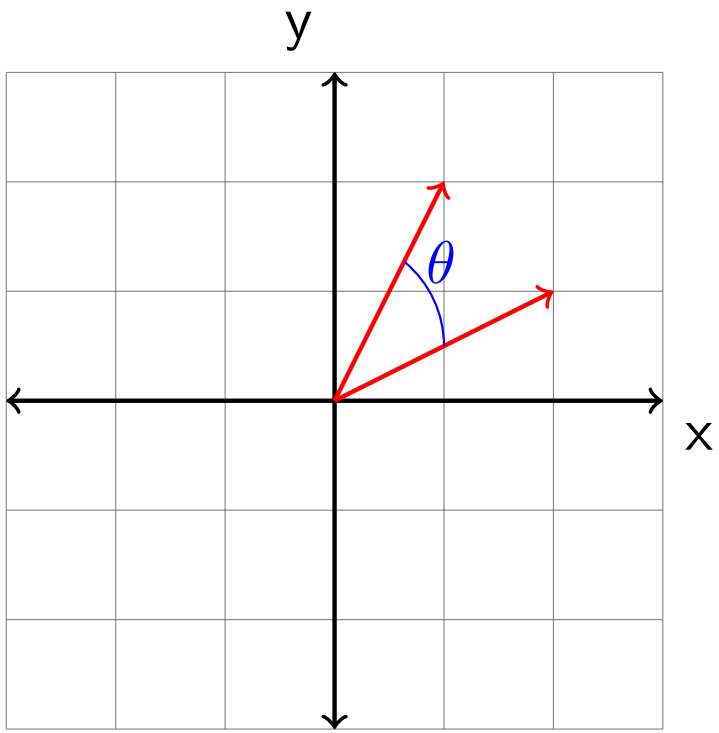


# Code for inner product

```
float in(struct vec v1, struct vec v2) {
····return v1.x * v2.x + v1.y * v2.y;
}
```

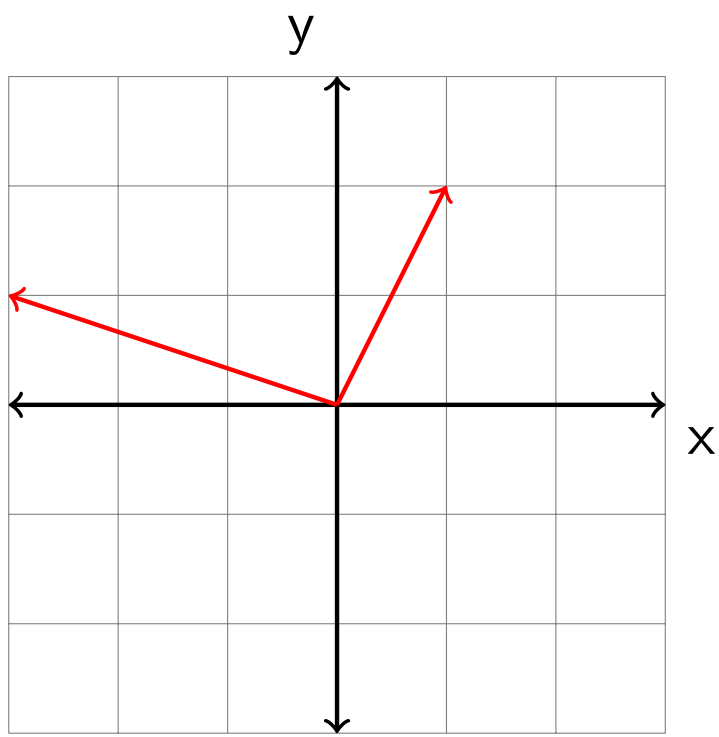
# Vectored

# Movement



# Matrix, representation

$$\mathbf{M} = \begin{bmatrix} 0.2 & 0.3 \\ 0.5 & 0.6 \end{bmatrix}$$



# Multiplying matrix with a vector

$$\mathbf{M} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a * x + b * y \\ c * x + d * y \end{bmatrix}$$

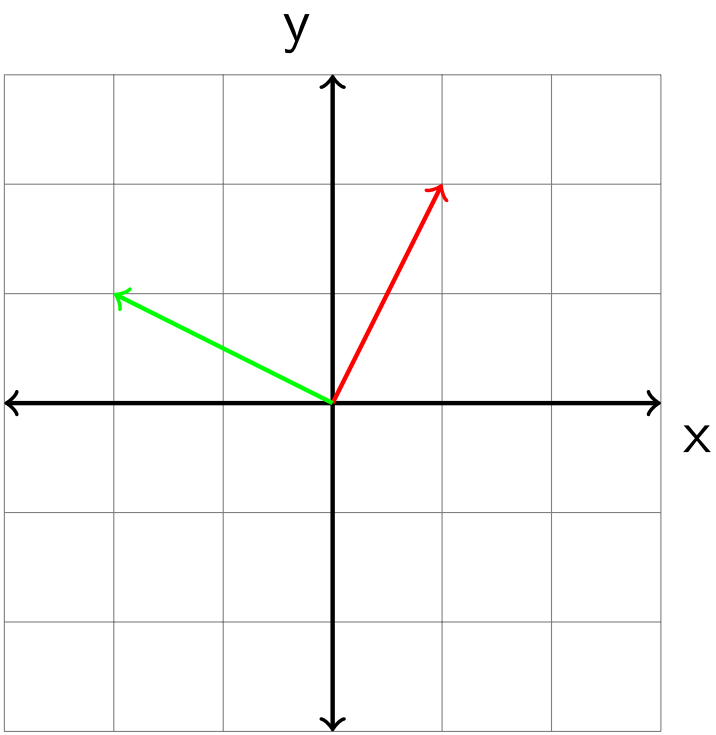
```
struct matrix {
    float a;
    float b;
    float c;
    float d;
};

struct vec matrix_mult(struct matrix m, struct vec v) {
    struct vec result;
    result.x = m.a * v.x + m.b * v.y;
    result.y = m.c * v.x + m.d * v.y;
    return result;
}
```

# Identity matrix

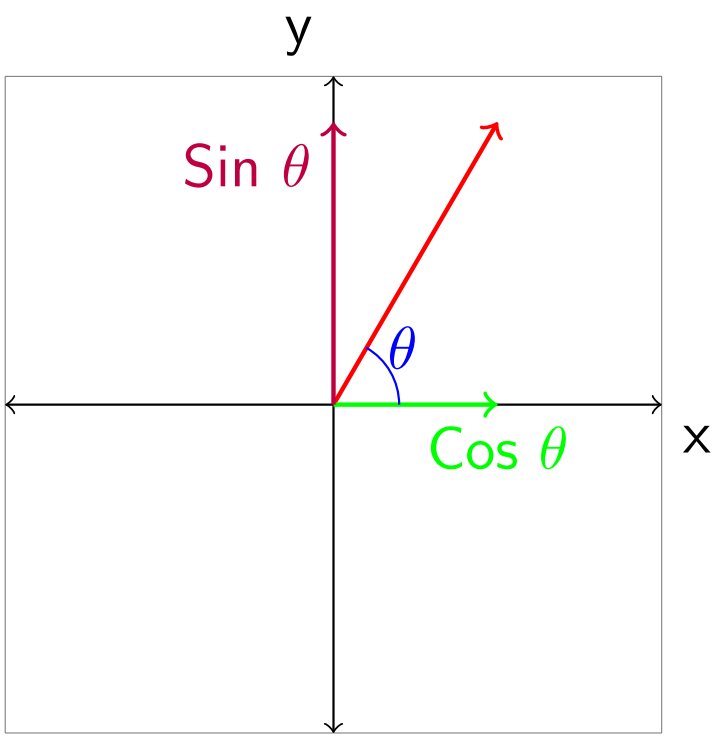
$$\mathbf{M} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} * \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 1*2+0*3 \\ 0*2+1*3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

# 90 degrees rotation matrix

$$\mathbf{M} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} * \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 * 1 + -1 * 2 \\ 1 * 1 + 0 * 2 \end{bmatrix} = \begin{bmatrix} -2 \\ 1 \end{bmatrix}$$

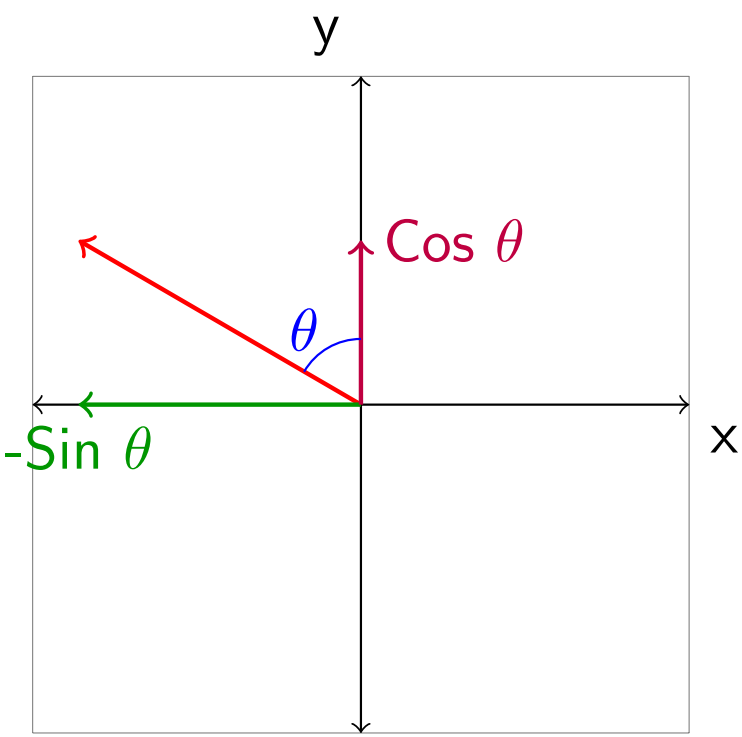


# Sin and Cos, rotation from x-axis




$$\mathbf{M} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \end{bmatrix} * \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}$$

# Sin and Cos, rotation from y-axis

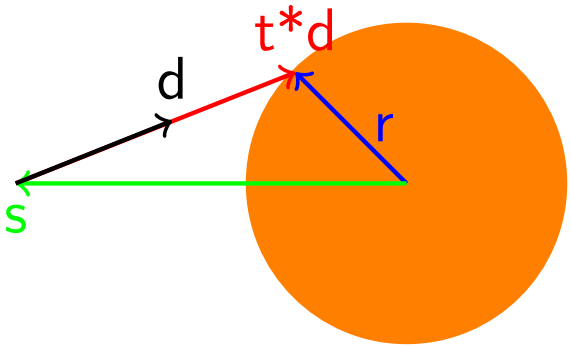



$$\mathbf{M} = \begin{bmatrix} 0 & -\sin(\theta) \\ 0 & \cos(\theta) \end{bmatrix} * \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -\sin(\theta) \\ \cos(\theta) \end{bmatrix}$$

# Rotation matrix

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

# Do we hit the orange




$$(a+b)^2 = a^2 + b^2 + 2 * a * b$$

$$(\vec{s} + t * \vec{d})^2 = r^2$$

$$t^2 * \vec{d}^2 + t * 2(\vec{s} \cdot \vec{d}) + \vec{s}^2 - r^2 = 0$$

# Quadratic formula

$$t^2 * a + t * b + c = 0$$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$b^2 - 4ac >= 0$$

# Quadratic formula

$$t^2 * \vec{d}^2 + t * 2(\vec{s} \cdot \vec{d}) + \vec{s}^2 - r^2 = 0$$

$$a = \vec{d}^2$$

$$b = 2(\vec{s} \cdot \vec{d})$$

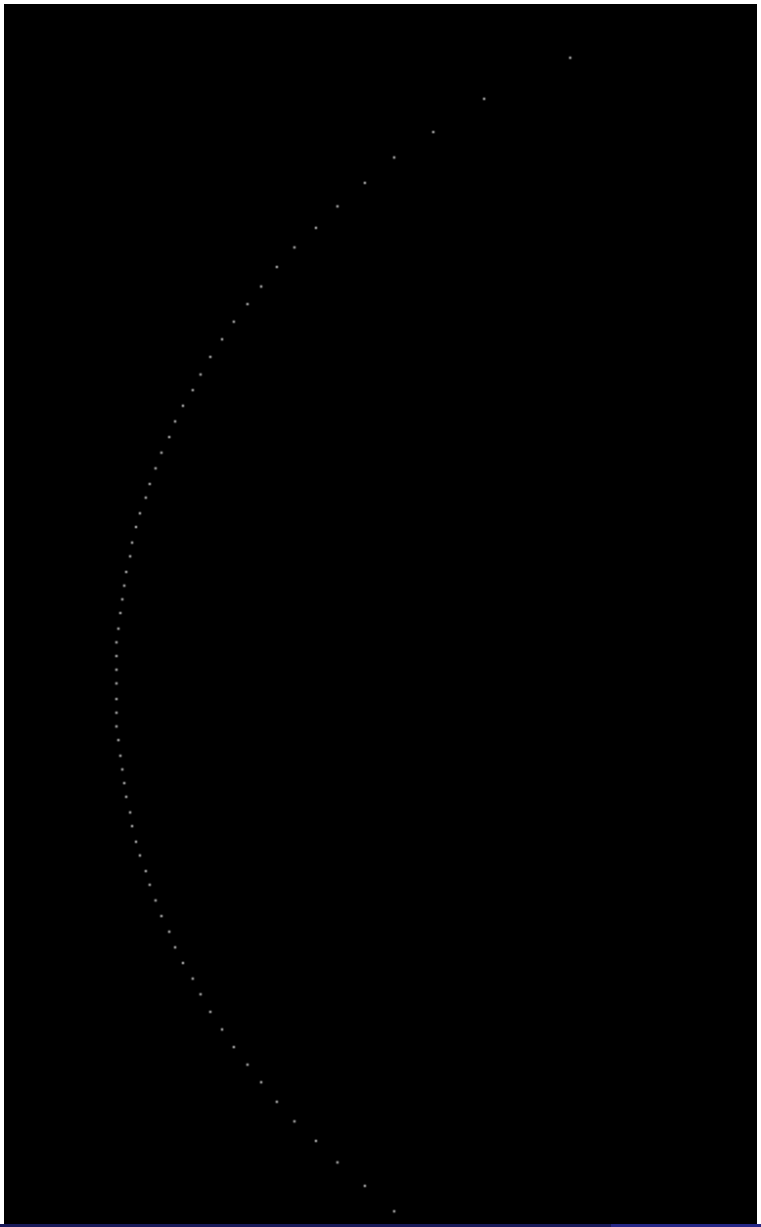
$$c = \vec{s}^2 - r^2$$

$$b^2 - 4ac >= 0$$

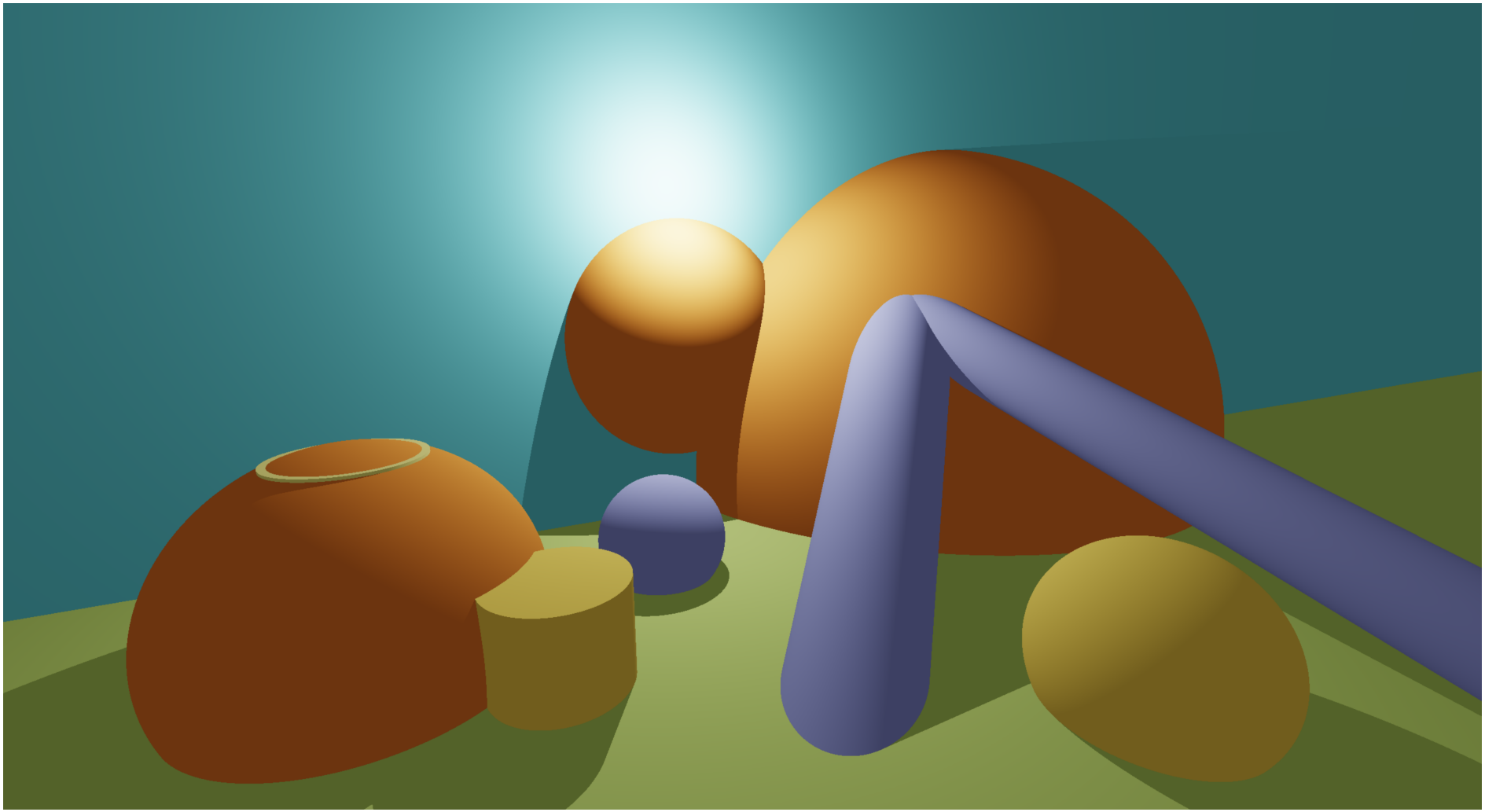
# Code for hitting the circle

```
for (float y = -2; y < 2; y += 0.01) {
    struct vec s = {-3, 0};
    struct vec d = {1, y};
    float r = 1;
    float a = in(d, d);
    float b = 2 * in(s,d);
    float c = in(s, s) - r*r;
    if (b*b - 4 * a * c > 0) {
        float t = (-b - (sqrt(b*b - 4*a*c))) / (2 * a);
        struct vec td = scale(d, t);
        pixel_put_image(&(info.img), SCREEN_WIDTH / 2 + td.x*(SCREEN_WIDTH /3),
                SCREEN_HEIGHT /2 + td.y * (SCREEN_HEIGHT /2 ), 0x00ffffff);
    }
}
```

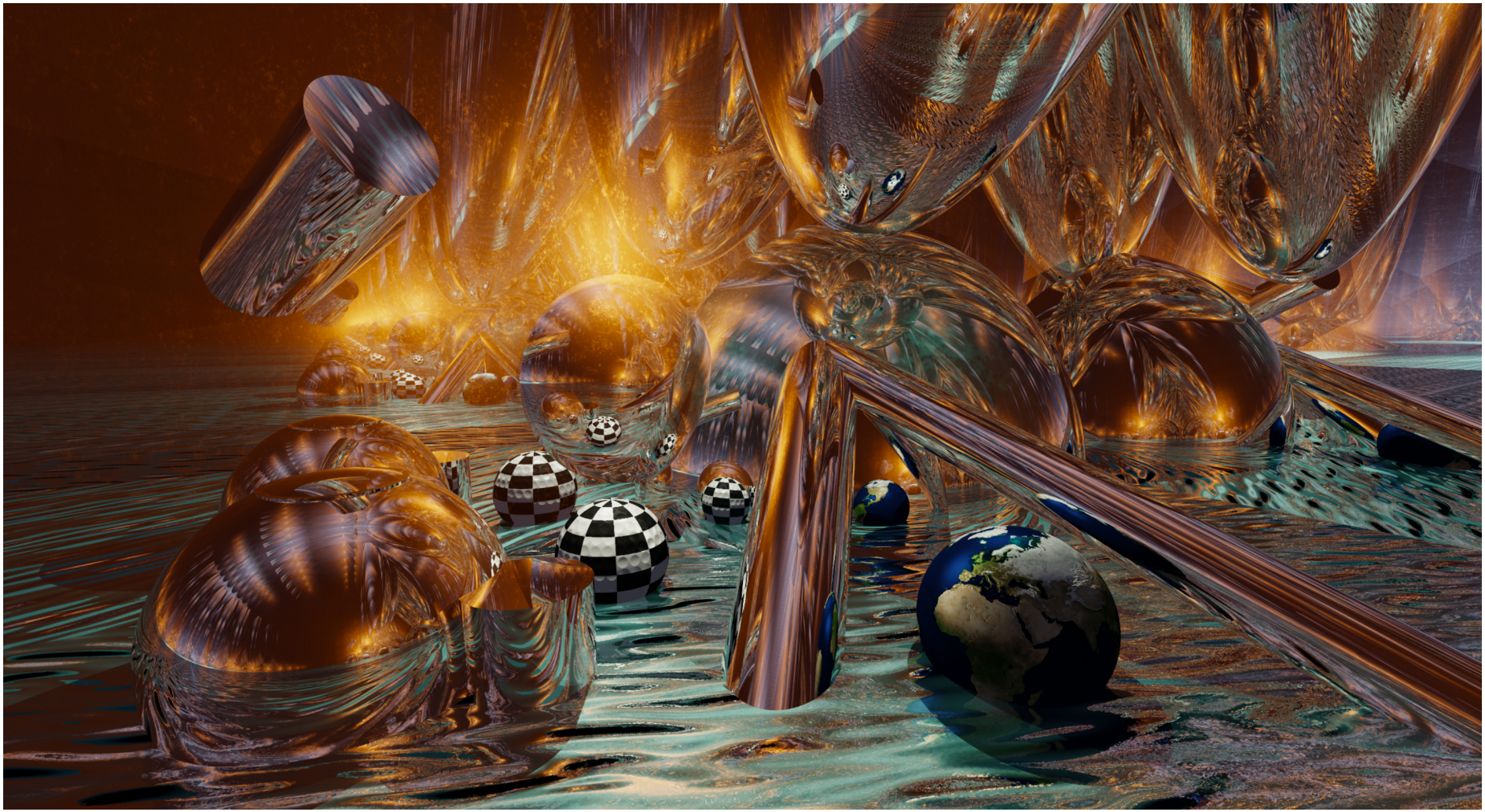
# Result of hitting the circle



# 3D version



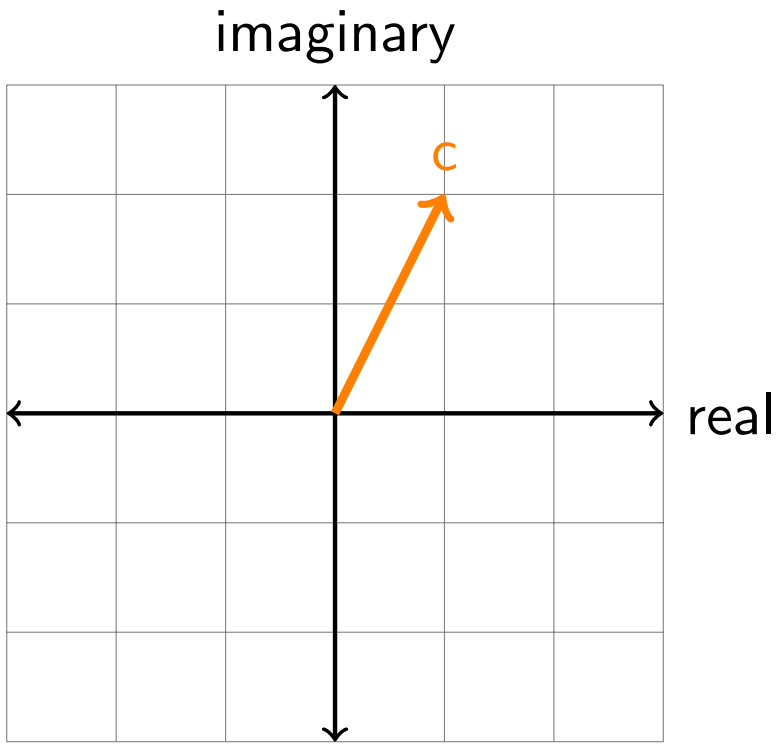
# And then...



# Complex numbers

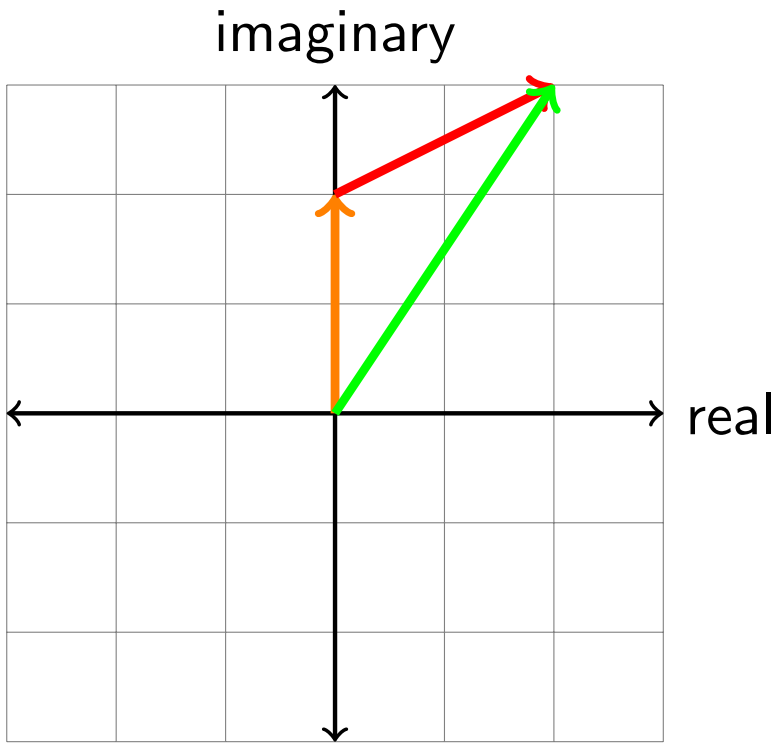
$$c = a + bi$$

$$i^2 = -1$$



# Adding complex numbers

$$(0 + 2i) + (2 + 1i) = 2 + 3i$$



# Code for adding complex numbers

```
struct cpx {
    float r;
    float i;
};

struct cpx add_cpx(struct cpx c1, struct cpx c2) {
    c1.r += c2.r;
    c1.i += c2.i;
    return c1;
}
```

# Multiplying complex numbers

$$(a_1 + b_1 i) * (a_2 + b_2 i) = (a_1 * a_2) + (a_1 * b_2 i) + (b_1 i * a_2) + (b_1 i * b_2 i)$$

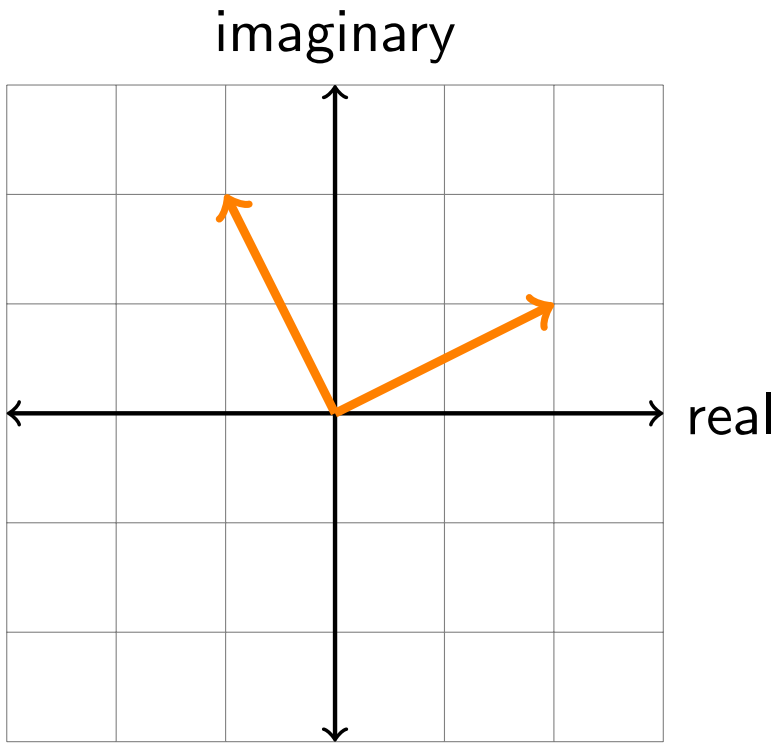
$$= (a_1 * a_2 - b_1 * b_2) + (a_1 * b_2 + b_1 * a_2) i$$

# Code for multiplying complex numbers

```
struct cpx mult_cpx(struct cpx c1, struct cpx c2) {
····float real = (c1.r * c2.r) - (c1.i * c2.i);
····float imag = (c1.r * c2.i) + (c1.i * c2.r);
····return (struct cpx) {real, imag};
}
```

# Multiplying complex numbers example

$$i * (2 + 1i) = -1 + 2i$$



# Complex numbers, rotation

$$(\cos(\theta) + i * \sin(\theta)) * (a + bi)$$

$$= \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} * \begin{bmatrix} a \\ b \end{bmatrix}$$

# Code for taking the squared length of complex numbers

```
float length_squared(struct cpx c) {
····return c.r * c.r + c.i * c.i;
}
```

# Julia set, Mandelbrot set

$$z = z^2 + c$$

# Mandelbrot

$$z = z^2 + c$$

$$1 = 0^2 + 1$$
$$2 = 1^2 + 1$$
$$5 = 2^2 + 1$$

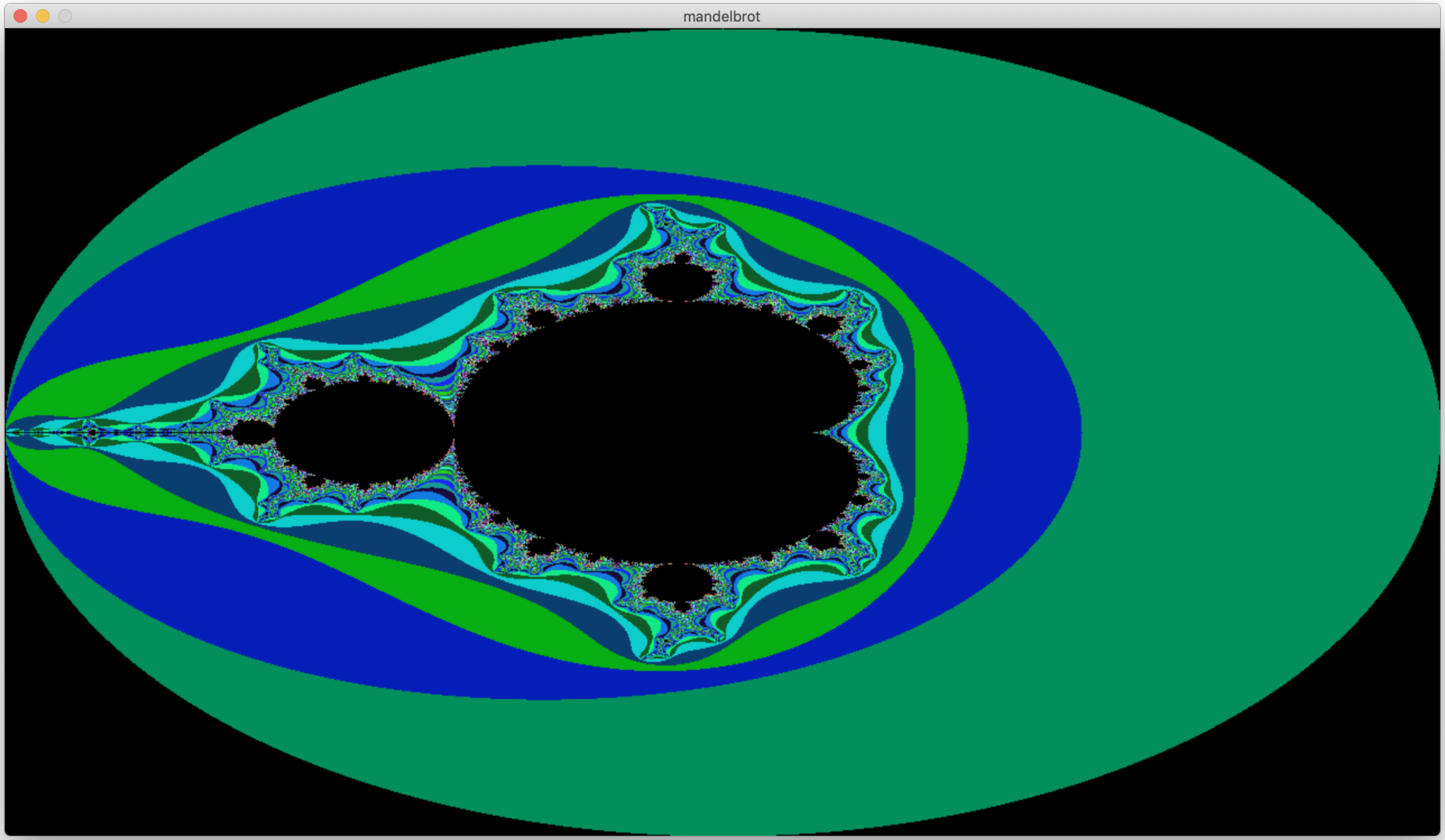
$$0 = 0^2 + 0$$

$$0.25 = 0^2 + 0.25$$
$$0.3125 = 0.25^2 + 0.25$$
$$0.3477 = 0.3125^2 + 0.25$$
$$\dots$$
$$0.25 = 0.5^2 + 0.25$$

# Mandelbrot code

```
  for (int i = 0; i < SCREEN_WIDTH; i++) {
      for (int j = 0; j < SCREEN_HEIGHT; j++) {
          int k = 0;
          int out = 0;
          struct cpx c = {(float)(i - SCREEN_WIDTH / 2) / SCREEN_WIDTH * 4,
                          (float)(j - SCREEN_HEIGHT / 2) / SCREEN_WIDTH * 4};
          struct cpx z = {0,0};
          for (k = 0; k < 100; k++) {
              z = add_cpx(mult_cpx(z, z), c);
              if (length_squared(z) > 4) {
                  out = 1;
                  break;
              }
          }
          if (out)
              pixel_put_image(&(info.img), i, j, (float) k / 100 * 0x00ffffff);
      }
  }
```

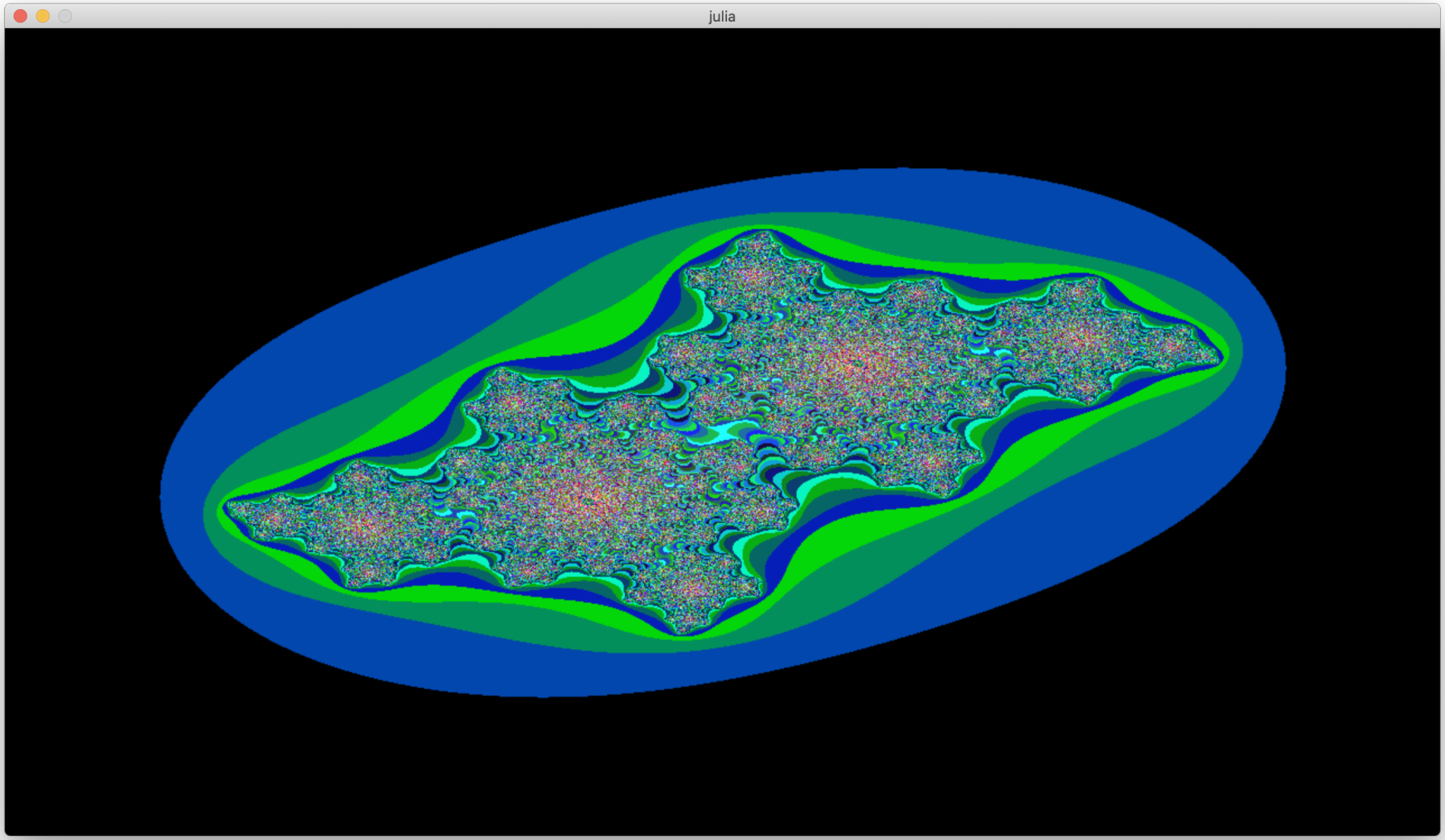
# Mandelbrot result



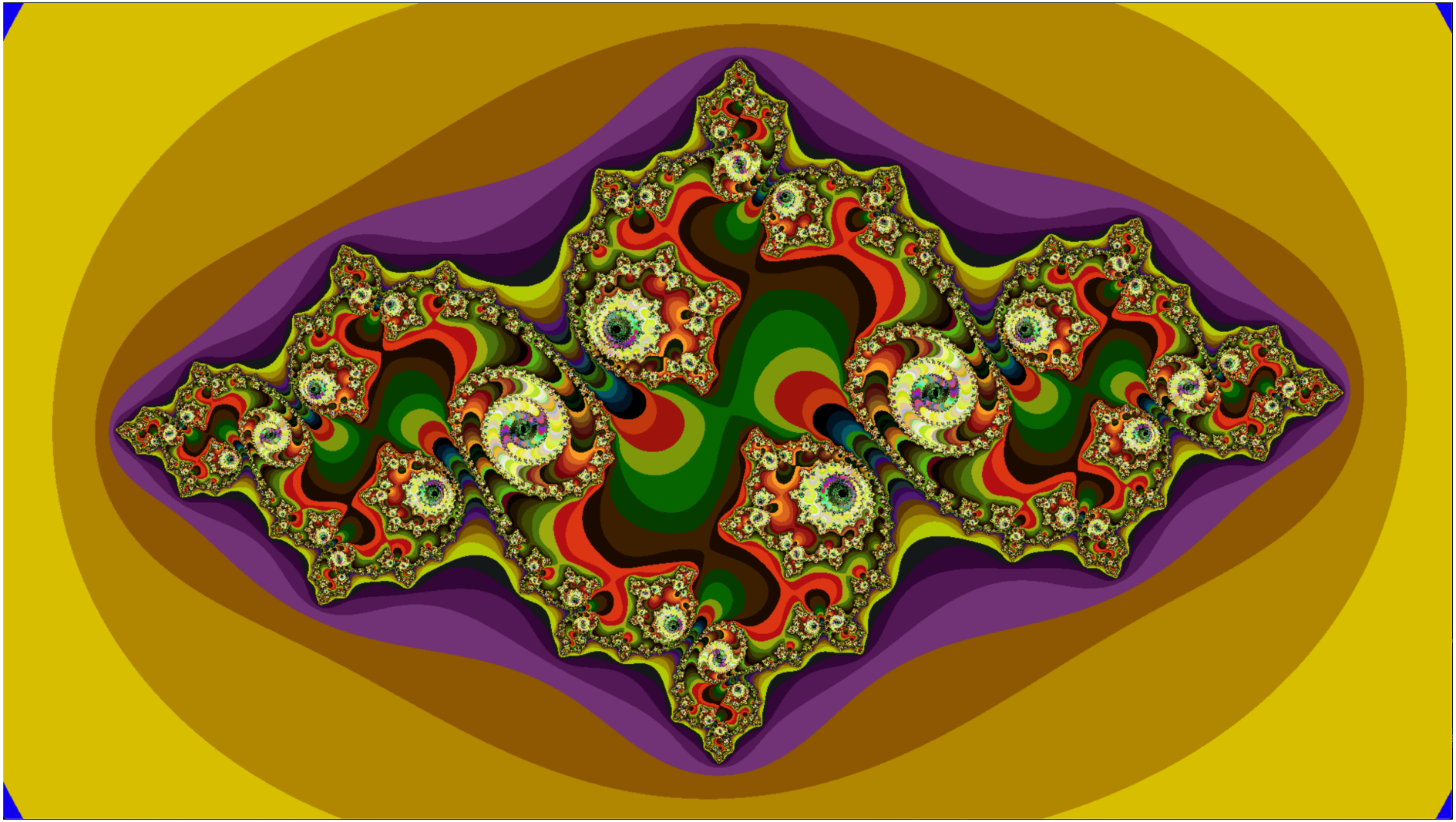
# Julia code

```
for (int i = 0; i < SCREEN_WIDTH; i++) {
    for (int j = 0; j < SCREEN_HEIGHT; j++) {
        int k = 0;
        int out = 0;
        struct cpx z = {(float)(i - SCREEN_WIDTH / 2) / SCREEN_WIDTH * 4,
                        (float)(j - SCREEN_HEIGHT / 2) / SCREEN_WIDTH * 4};
        struct cpx c = {-0.4,0.6};
        for (k = 0; k < 1000; k++) {
            z = add_cpx(mult_cpx(z, z), c);
            if (length_squared(z) > 4) {
                out = 1;
                break;
            }
        }
        if (out)
            pixel_put_image(&(info.img), i, j, (float) k / 200 * 0x00ffffff);
    }
}
```

# Julia result



# rvan-mee



# slides

https://github.com/GroteGnoom/codam_presentation