Computer Graphics

Lecture 5

Images, Rays,
and Cameras

or: I asked for an image and all I got was this grid of colored blocks

Announcements

Announcements

- Extra office hours 12:15-1:30 today.

Announcements

- Extra office hours 12:15-1:30 today.
- A1: how's it going?

Announcements

- Extra office hours 12:15-1:30 today.
- A1: how's it going?
- A2 out Friday, due Wednesday 2/5

Announcements

- Extra office hours 12:15-1:30 today.
- A1: how's it going?
- A2 out Friday, due Wednesday 2/5
- HW1 (tenatively) out Friday

Where were we?

Pseudocode for 3D graphics:

```
Create a model of a scene
Render an image of the model
```

Where were we?

Pseudocode for 3D graphics:

```
Create a model of a scene
Render an image of the model
```

Two Rendering Algorithms

Two Rendering Algorithms

```
for each object in the scene {
  for each pixel in the image {
    if (object affects pixel) {
      do something
    }
  }
}
```

object order
or
rasterization

Two Rendering Algorithms

```
for each object in the scene {
  for each pixel in the image {
    if (object affects pixel) {
      do something
    }
  }
}
```

object order
or
rasterization

```
for each pixel in the image {
  for each object in the scene {
    if (object affects pixel) {
      do something
    }
  }
}
```

image order
or
ray tracing

Today

`Render an image of the model`

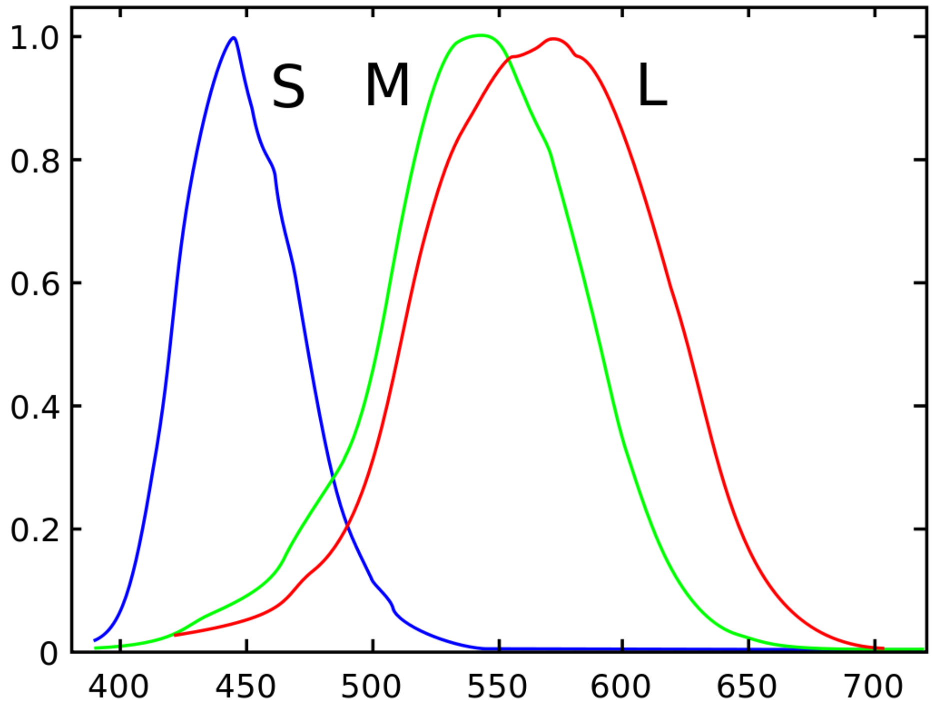
- What does image mean?
- What does render mean?
- Beginnings of **image-order** rendering (i.e., ray tracing)
  - Where do rays come from?

What is an image?

What is an image?

- At its most formal and general: a function mapping positions in 2D to distributions of radiant energy

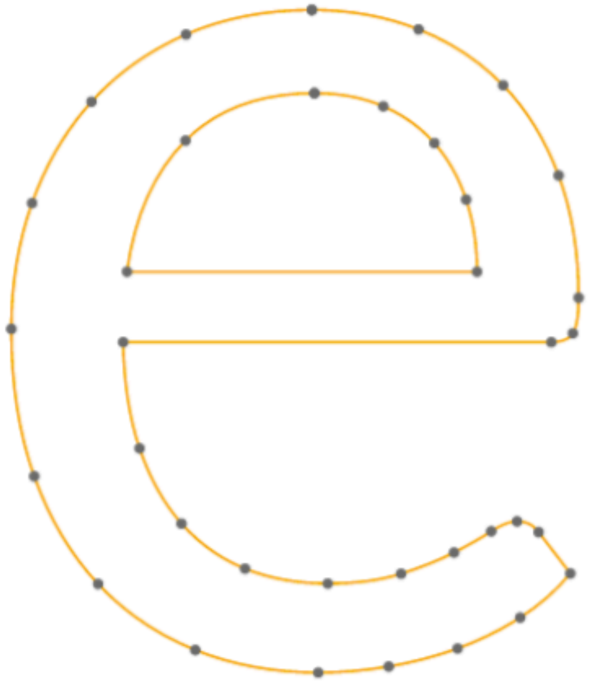
- Humans are trichromatic, so we usually represent color as combinations or red, green, and blue



How do we represent images?

- Raster formats - a 2D array of numbers
- Vector formats - mathematical description

Raster Image

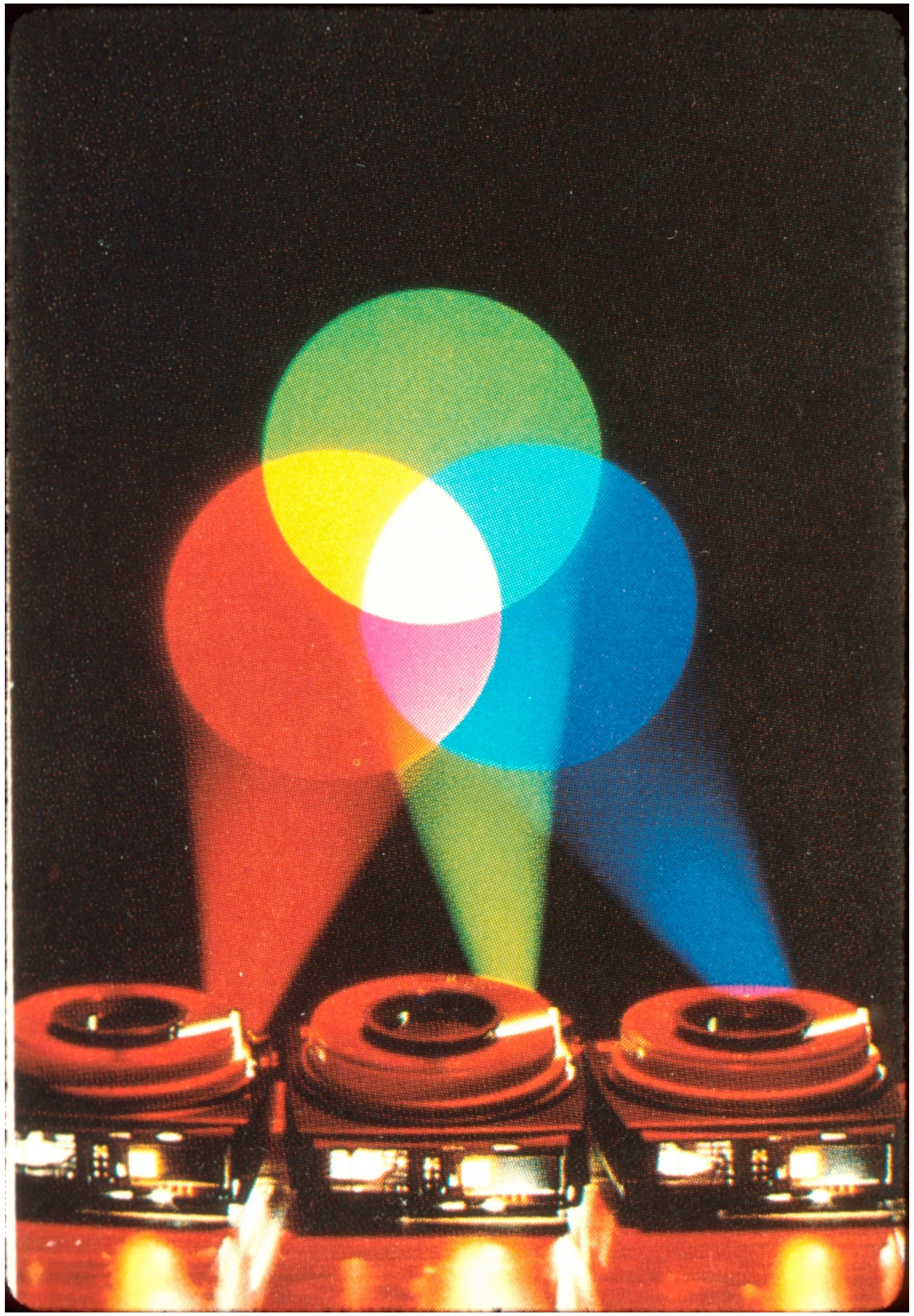


Pavithra Solai, kint.io

Vector Image

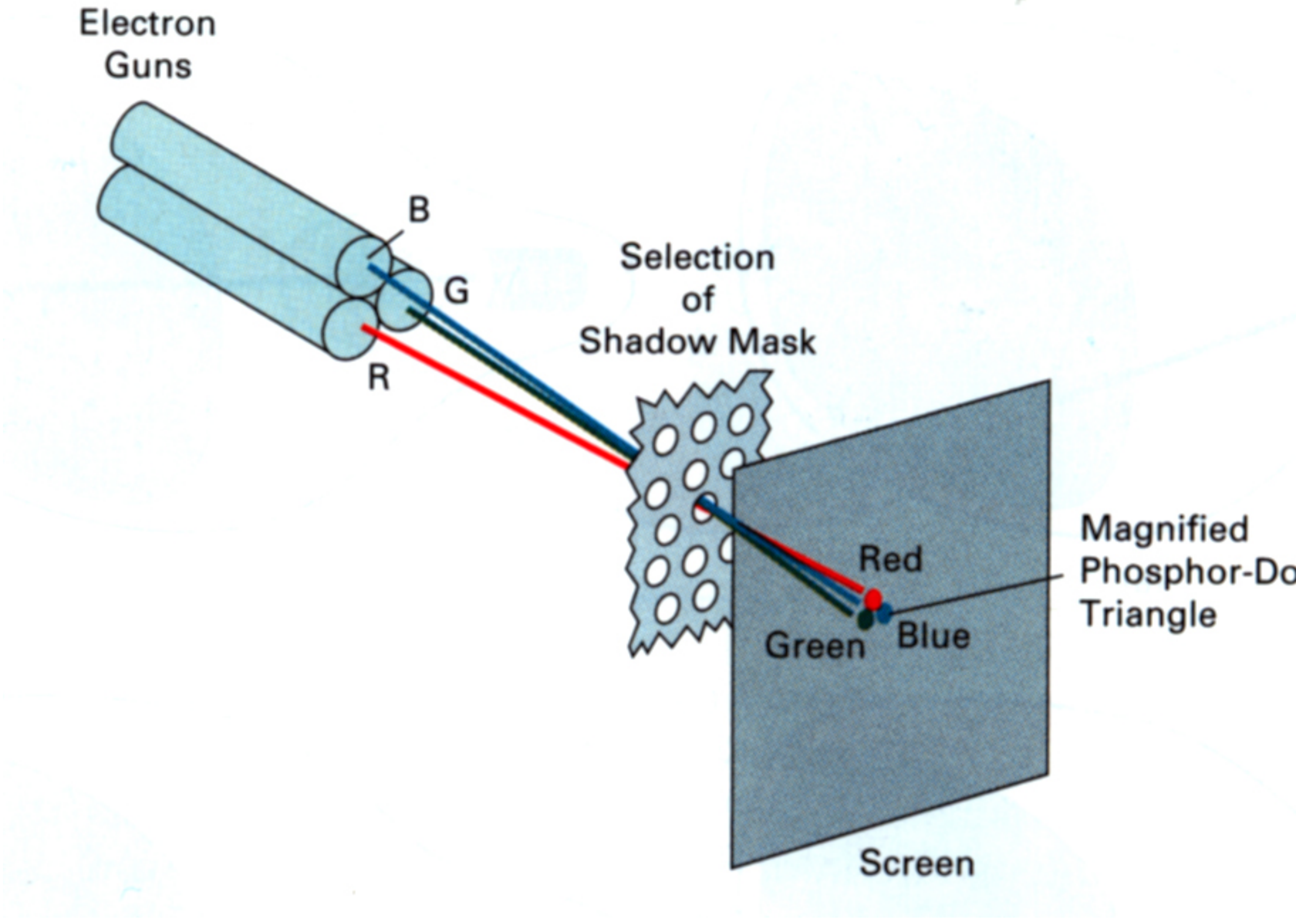
Color Displays - Old School

Color Projector



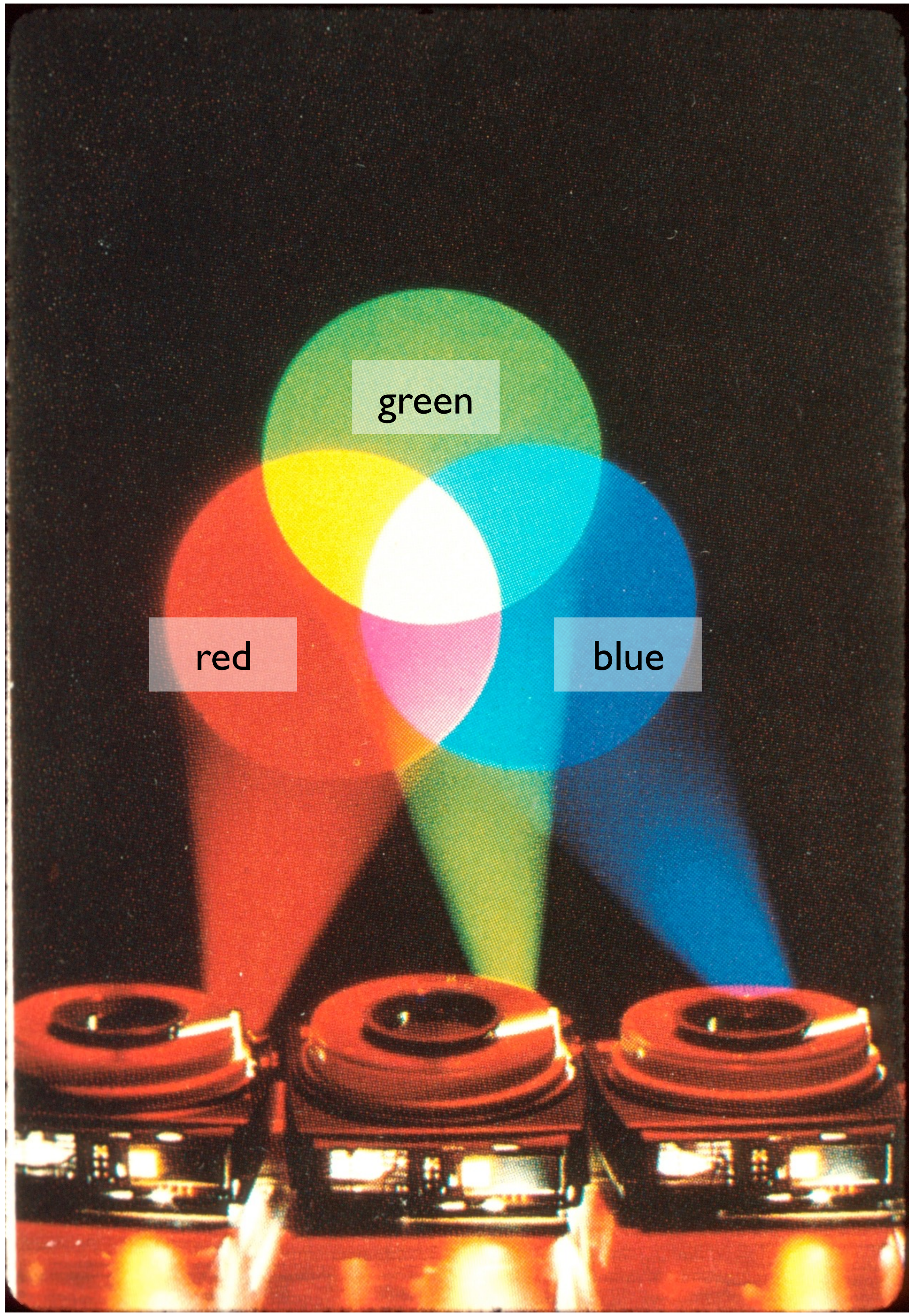
Cathode Ray Tube


Open CRT Monitor

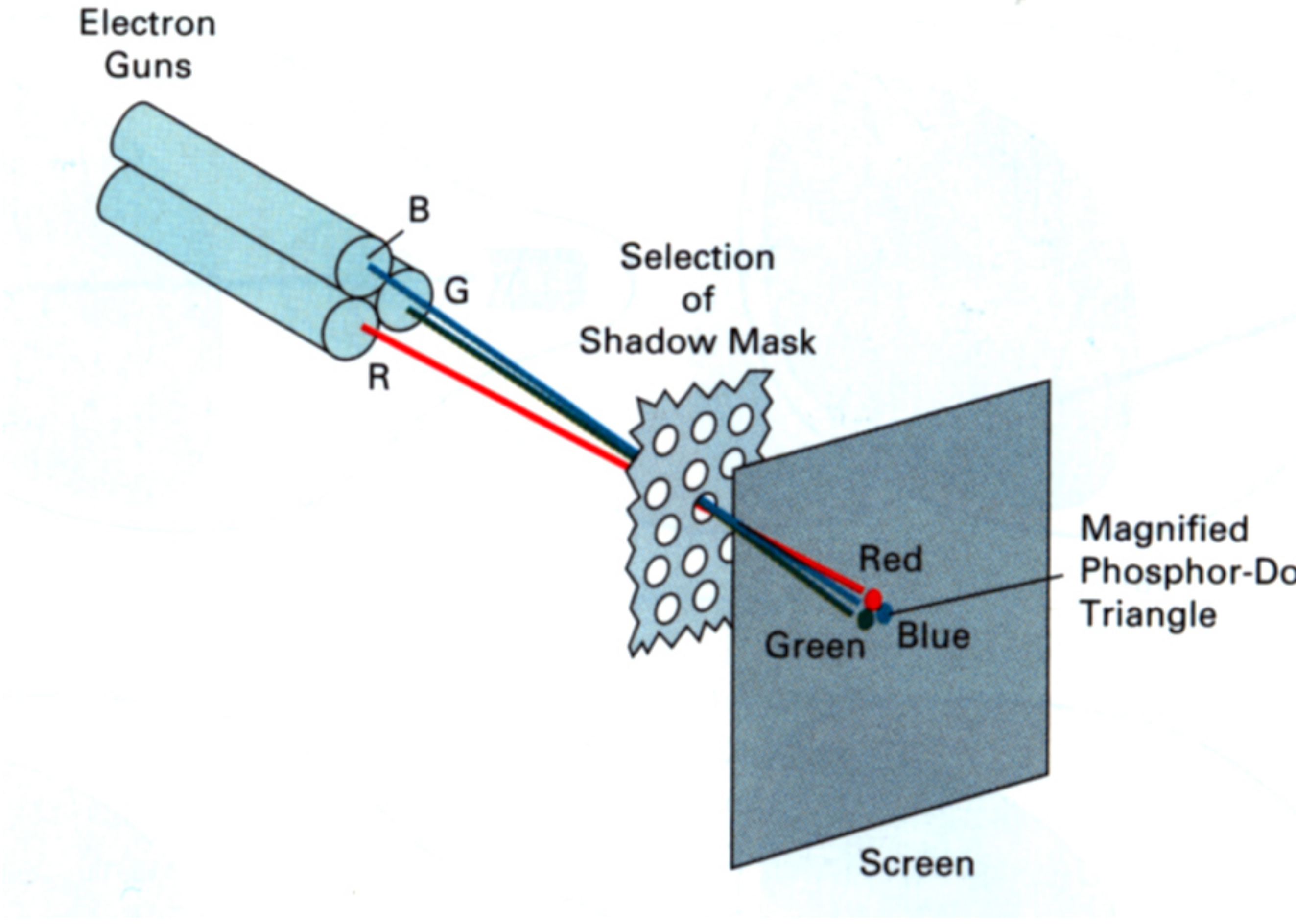


Color Displays - Old School

Color Projector

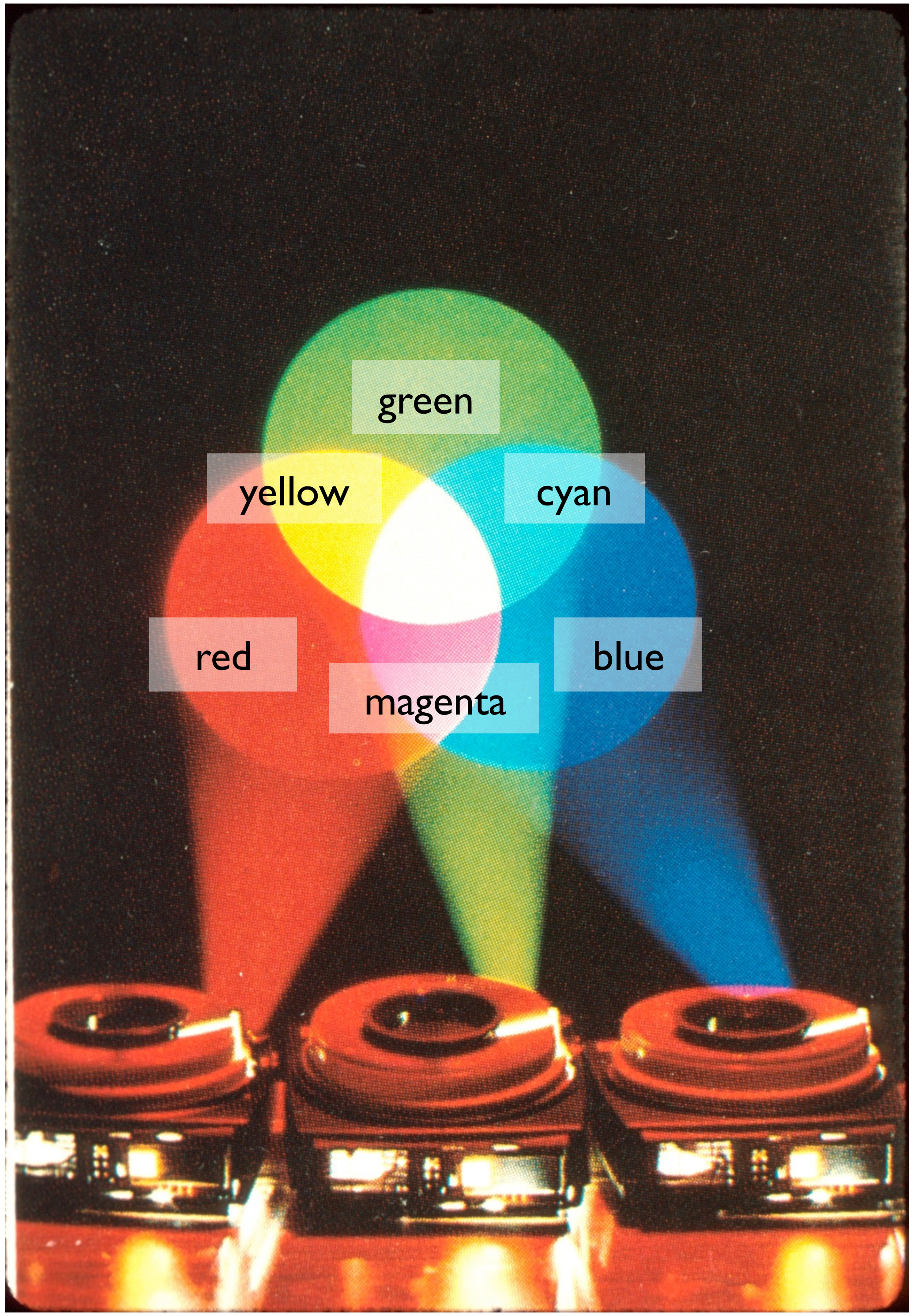


Cathode Ray Tube

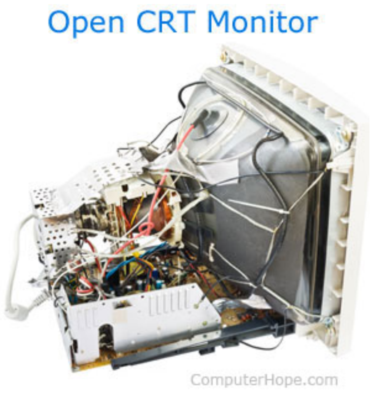


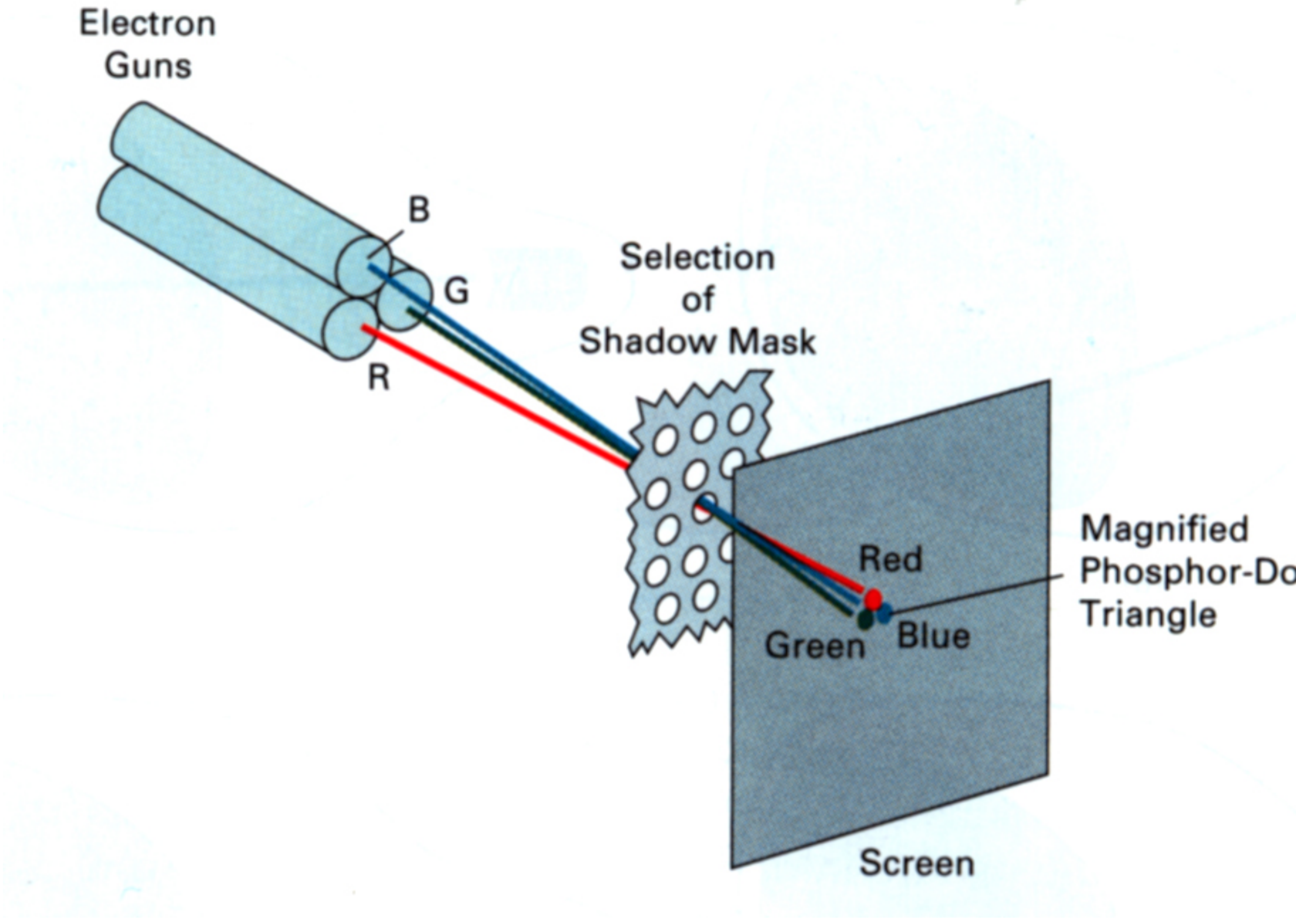
Color Displays - Old School

Color Projector



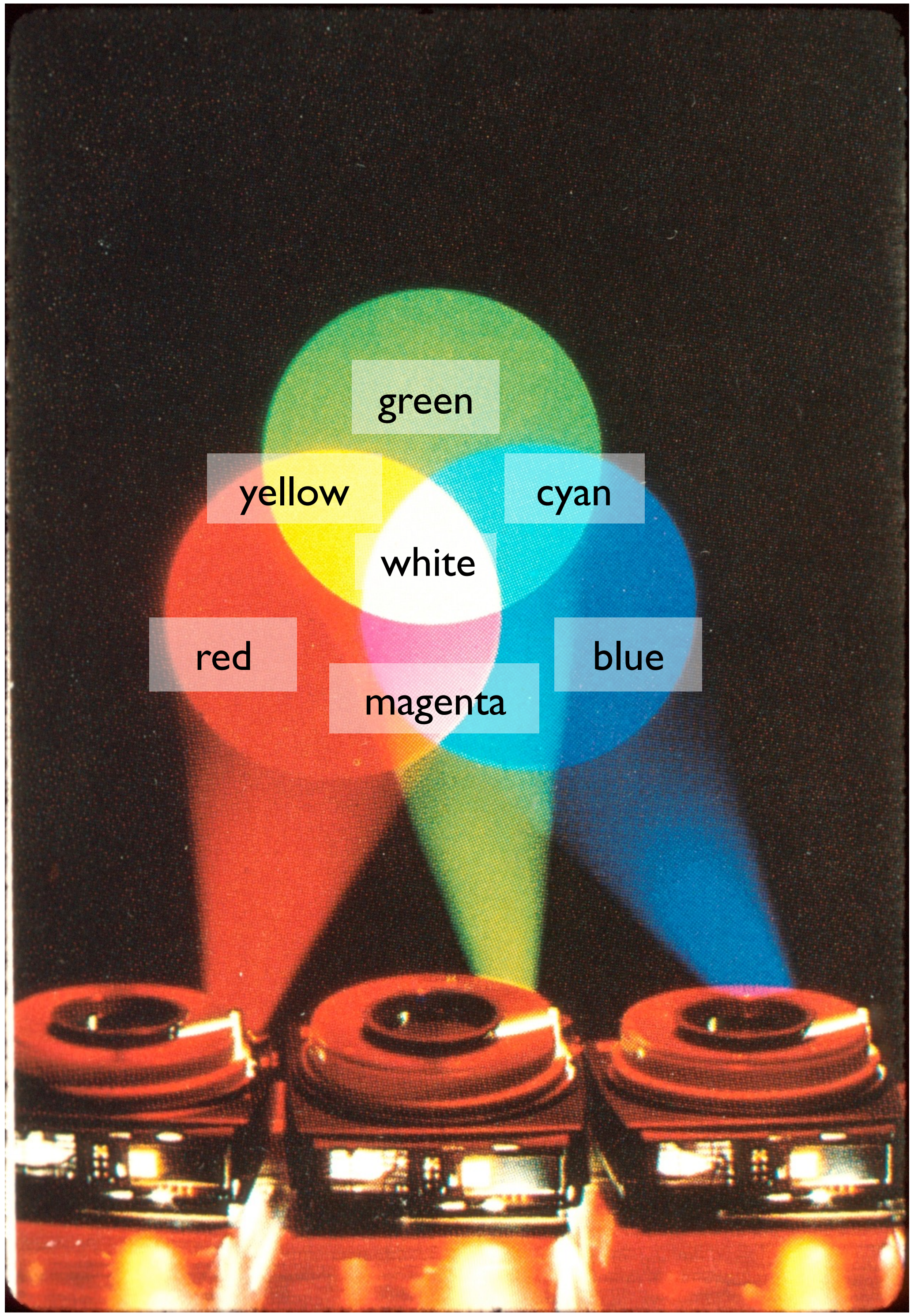
Cathode Ray Tube



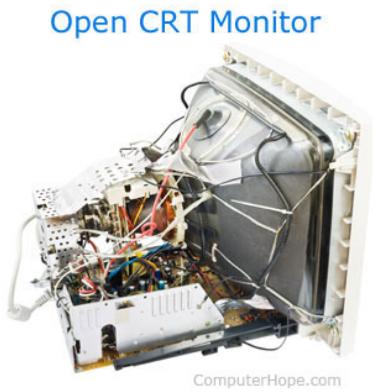


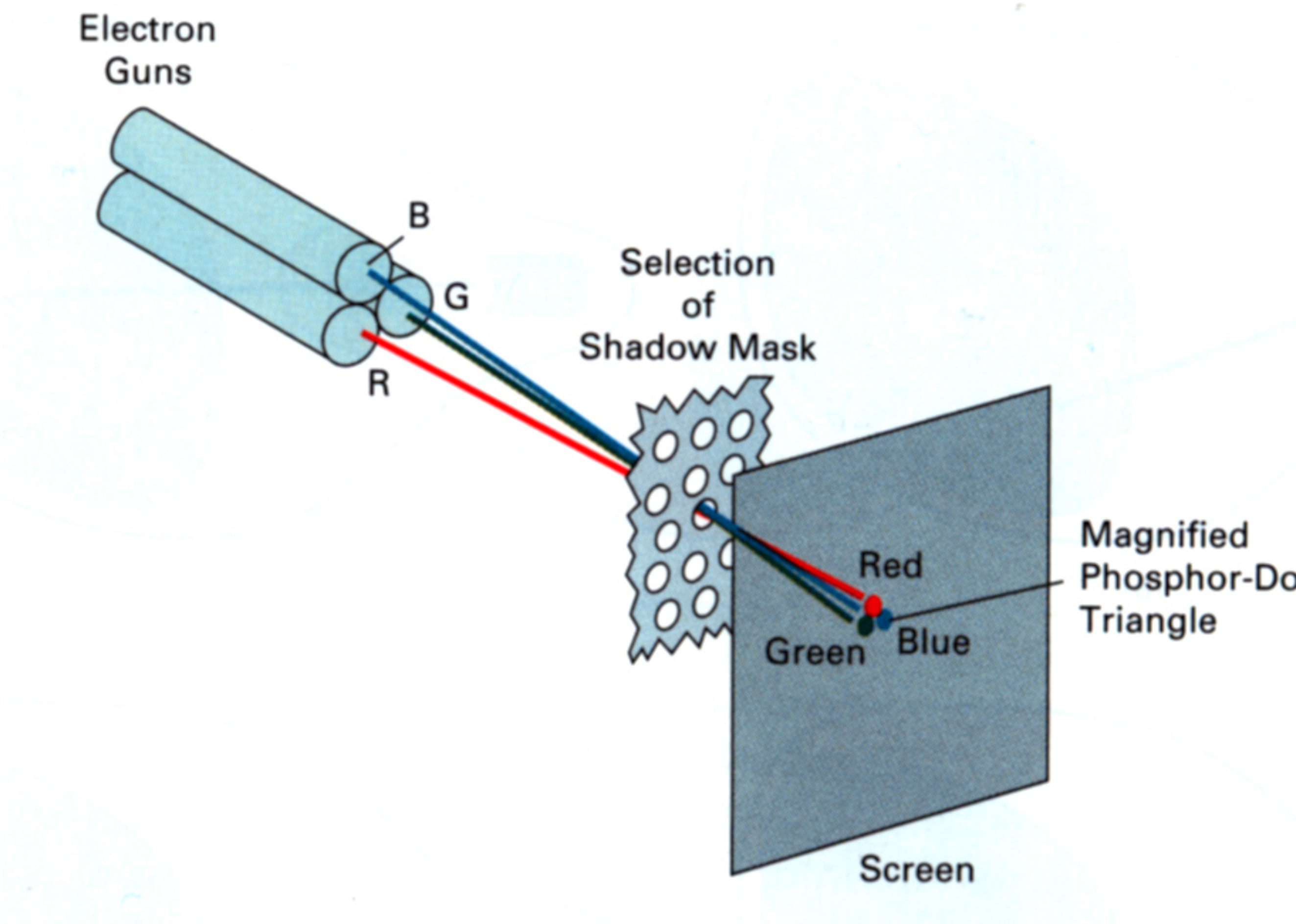
Color Displays - Old School

Color Projector



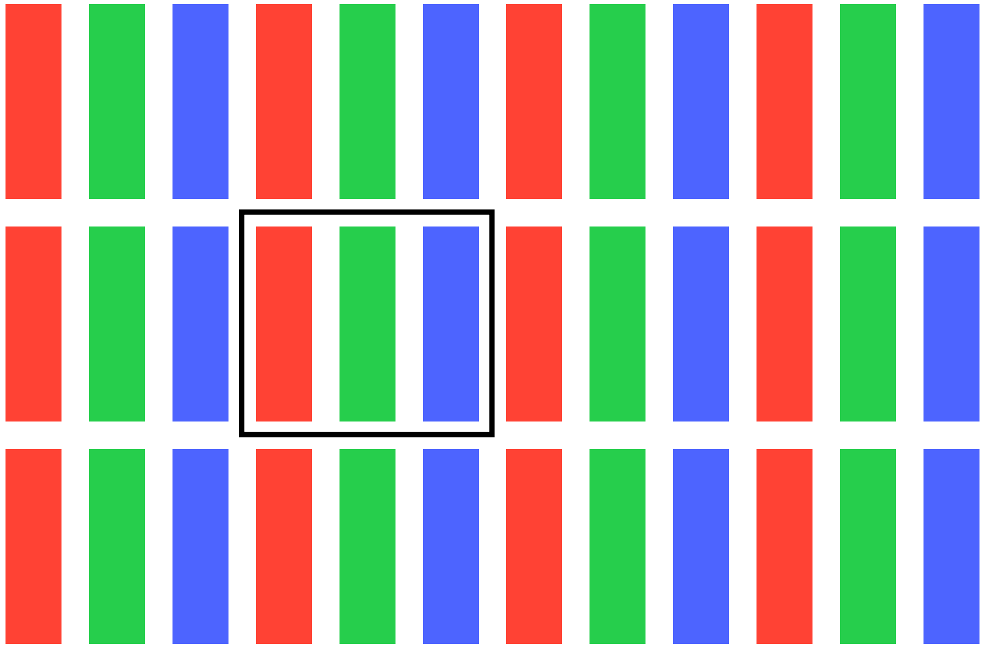
Cathode Ray Tube



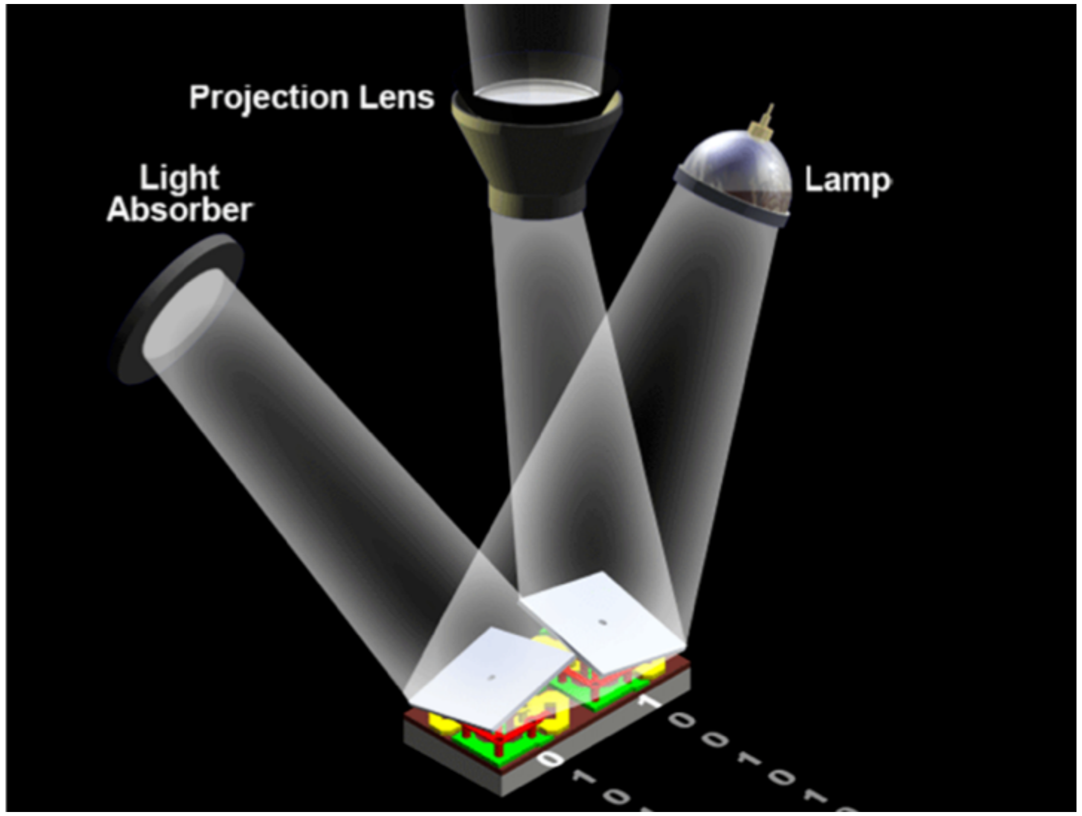


Color Displays - Nowadays

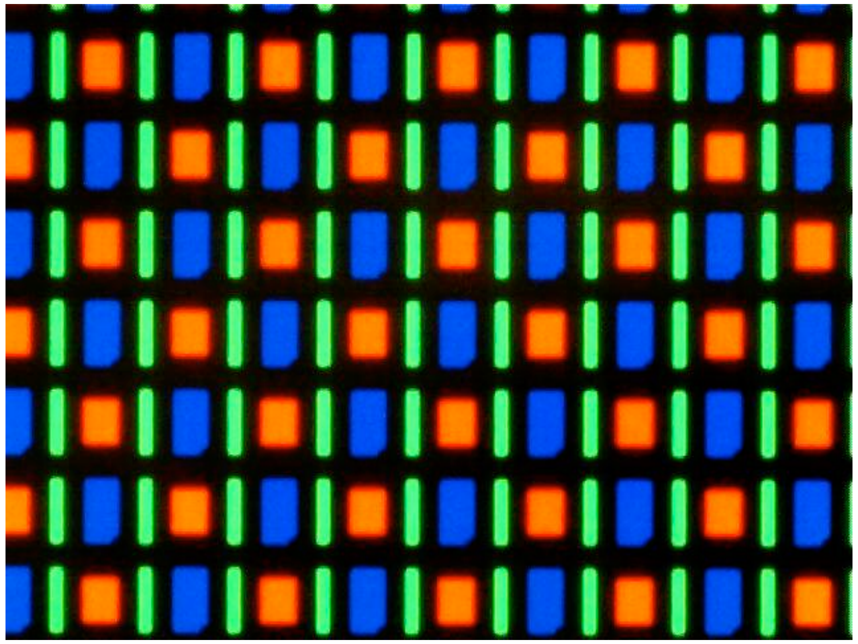
Liquid Crystal Display



Digital Light Processing



Light Emitting Diode Display



[Wikimedia Commons]

Raster Images

- Flexible
- Display-native
- Expensive

Raster Images:
2D Arrays of Numbers

- Bitmap (1 bit per pixel)
- Grayscale (usually 8 bpp)
- Color (usually 24 bpp)
- Floating-point (gray or color)
  - Bad for display, but good for processing
  - Allows **high dynamic range**

Raster Images: Storage

1 **megapixel** image - 1024x1024:

- Bitmap (1 bit per pixel) - **128 KB**
- Grayscale (8 bpp) - **1 MB**
- Color (24 bpp) - **3 MB**
- Floating-point (color) - **12MB**

2D Arrays in Julia

- A height-by-width array, each pixel is 3 single-precision floats initialized to zero:

```
canvas = zeros(RGB{Float32}, height, width)
```

2D Arrays in Julia

- A height-by-width array, each pixel is 3 single-precision floats initialized to zero:

```
canvas = zeros(RGB{Float32}, height, width)
```

```
canvas[i, j] # is the i'th row, j'th column
```

How do we make images?

How do we make images?

- IRL:
  - pencils, paintbrushes, watercolors, etc
  - eyes
  - **cameras**
- On computers:
  - virtual cameras

The Camera Conundrum:

The Camera Conundrum:

The world is 3D

The Camera Conundrum:

The world is 3D

Images are 2D

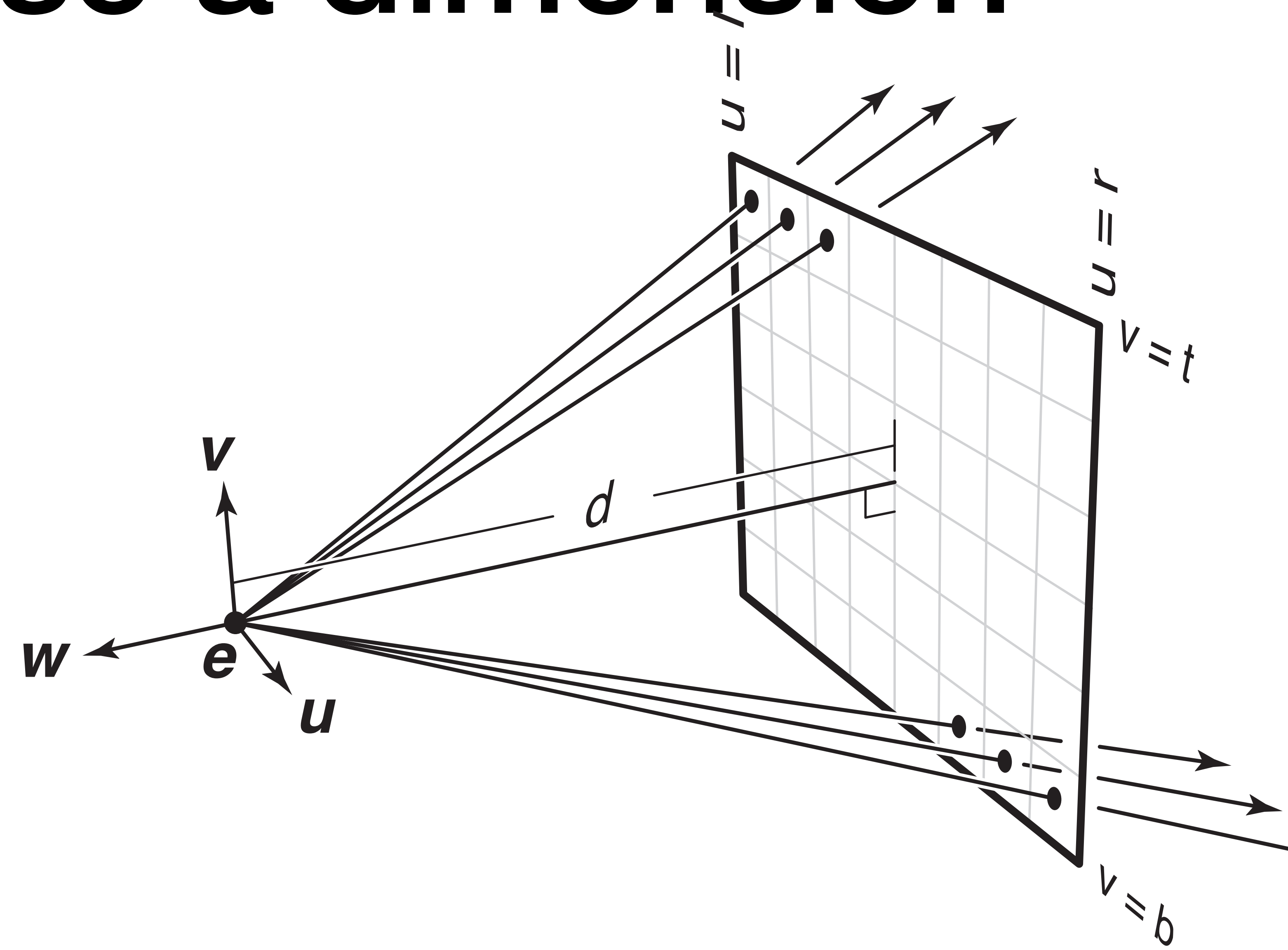
The Camera Conundrum:

The world is 3D

Images are 2D

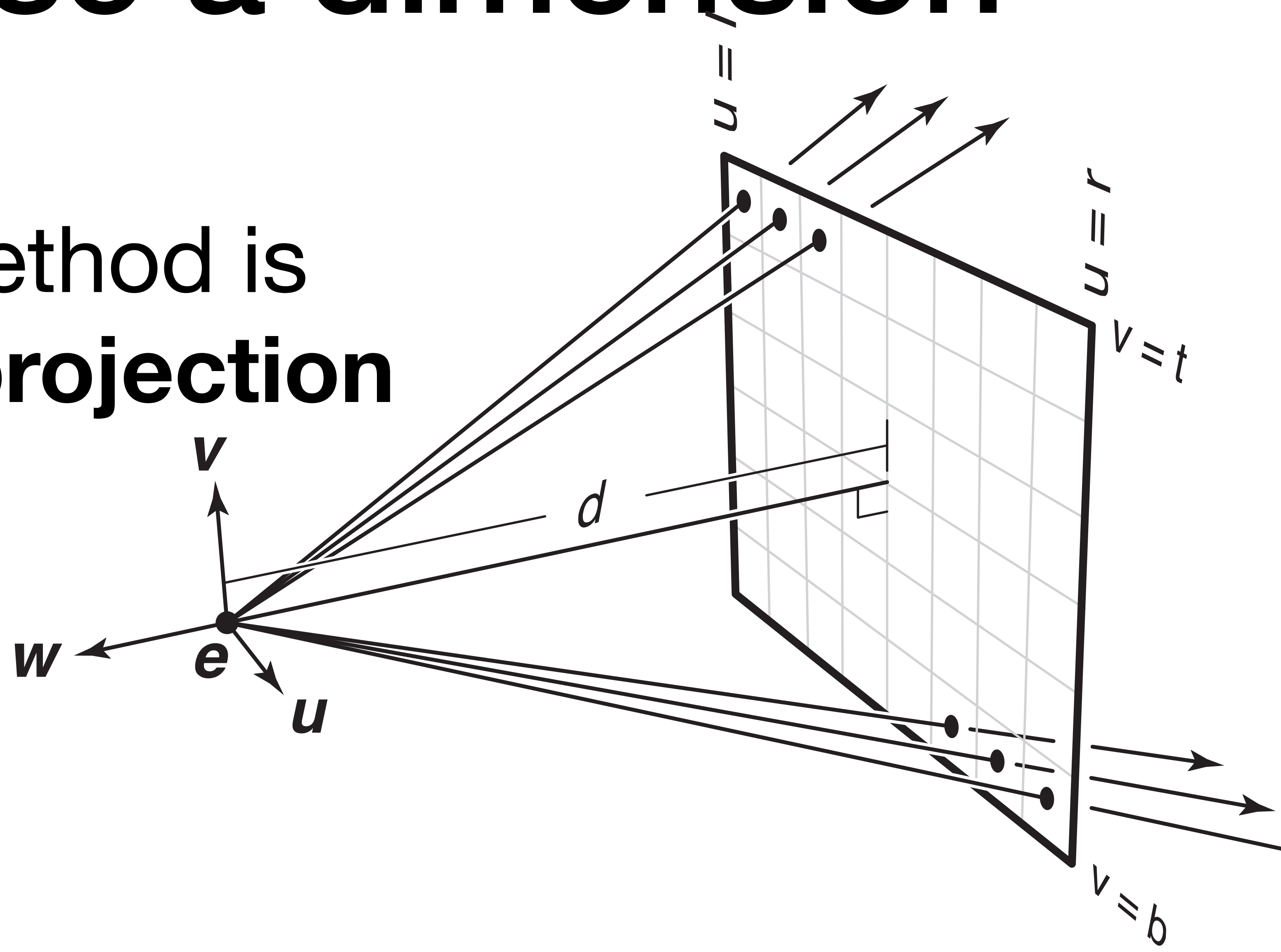
we gotta lose a dimension somehow

Projections:
ways to lose a dimension



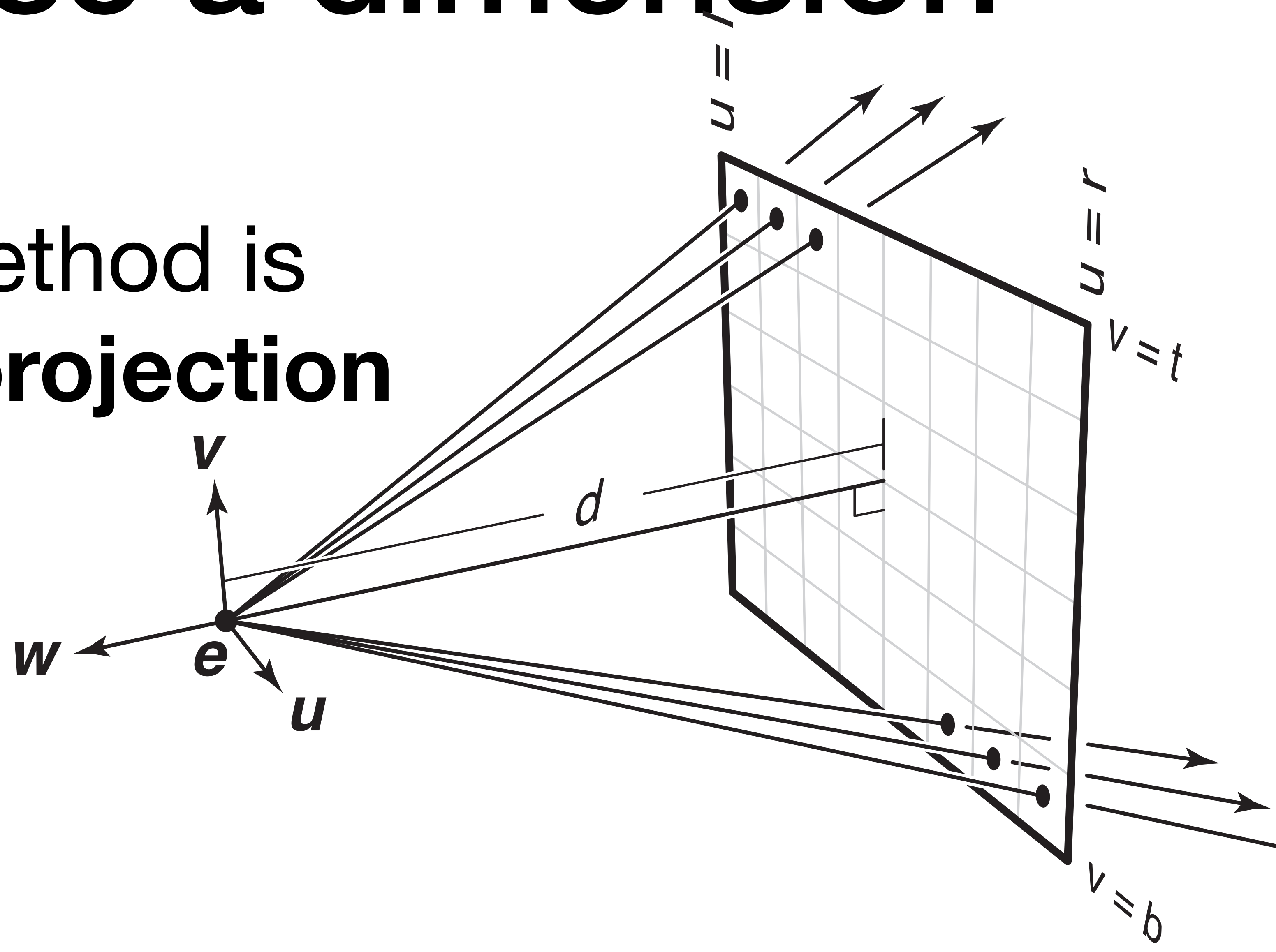
Projections: ways to lose a dimension

- The picture-frame method is called **perspective projection**



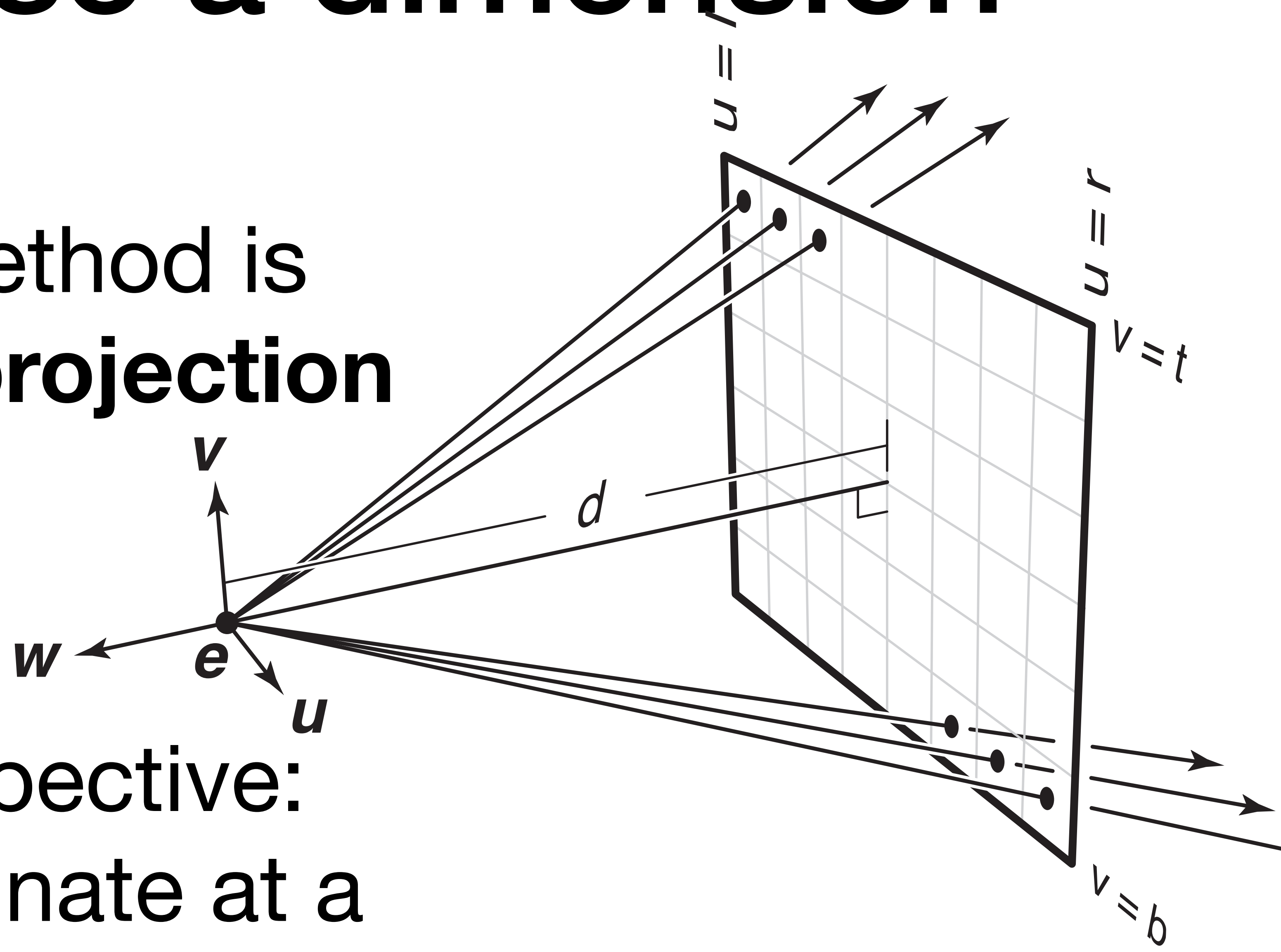
Projections: ways to lose a dimension

- The picture-frame method is called **perspective projection**



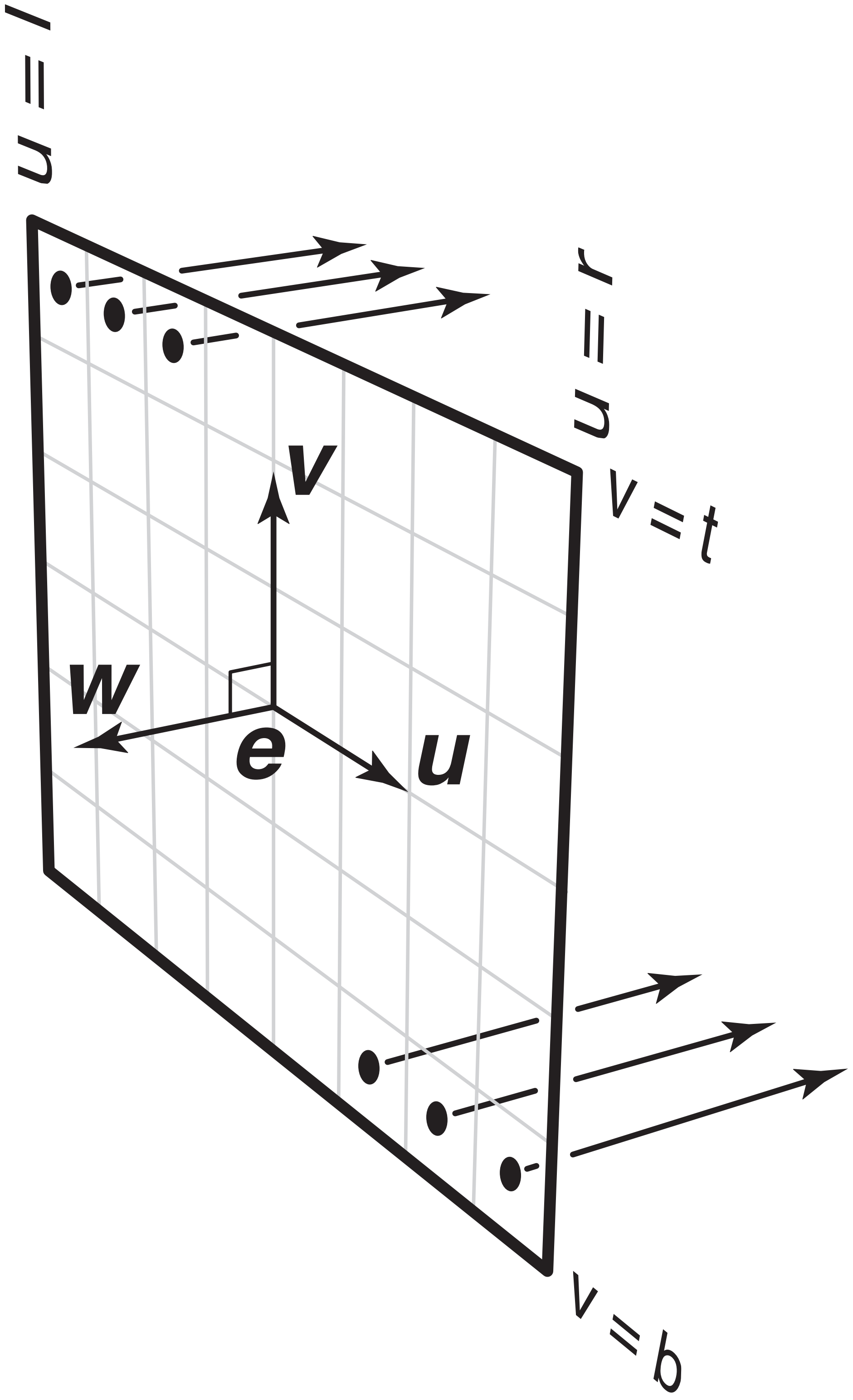
Projections:
ways to lose a dimension

- The picture-frame method is called **perspective projection**



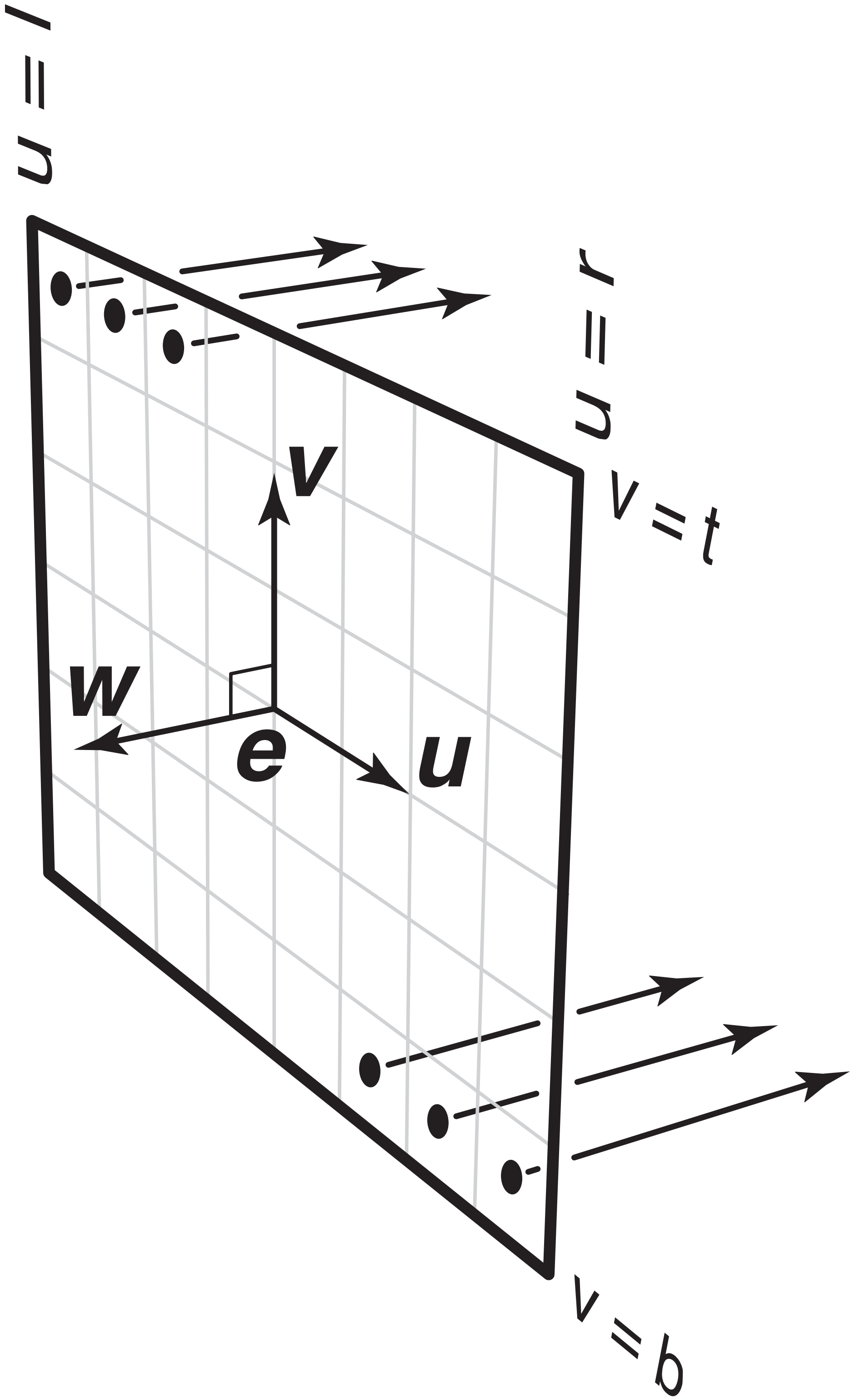
- Key property of perspective: all **viewing rays** originate at a single point, the *center of projection*, or *eye*.

Projections: ways to lose a dimension



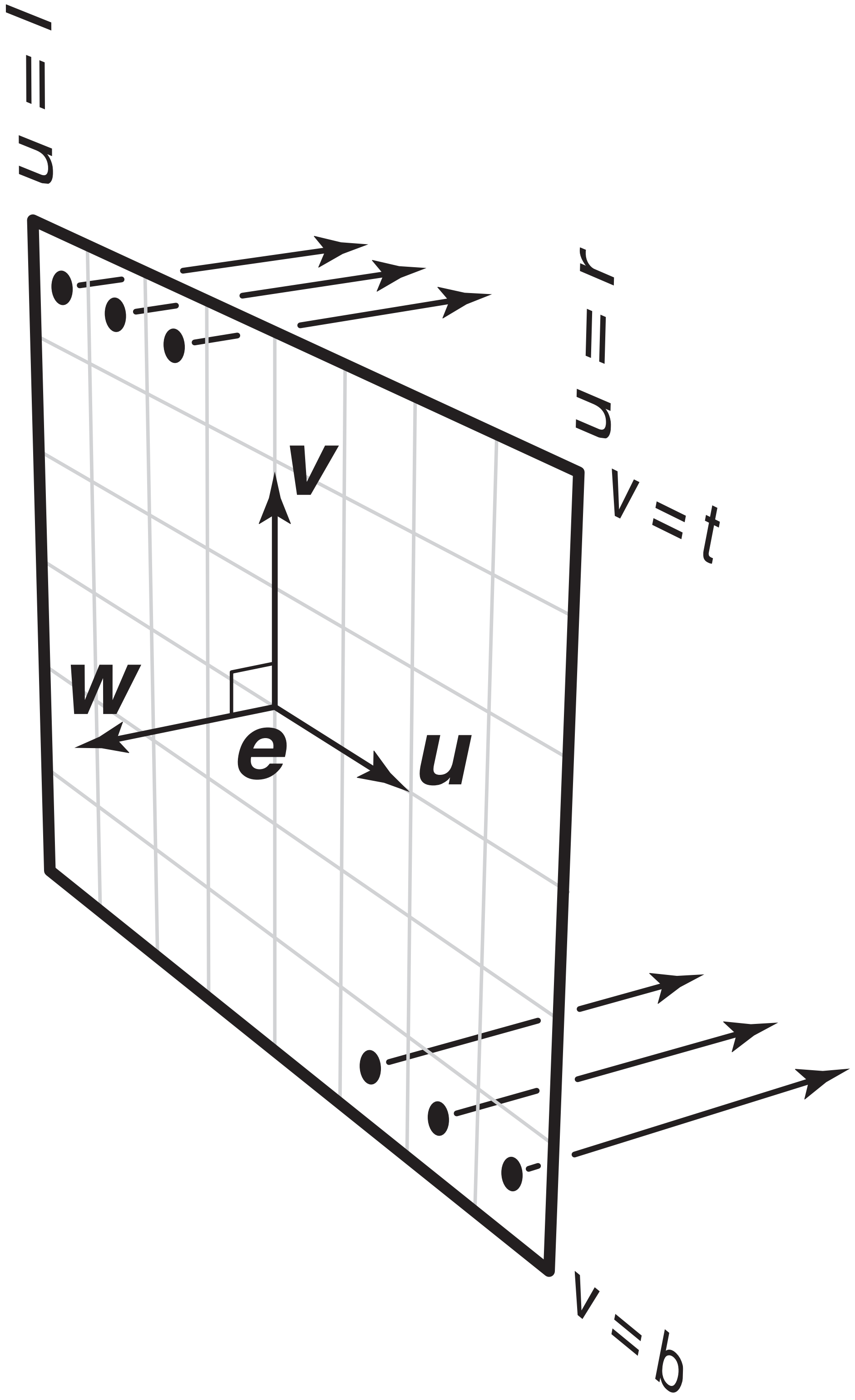
Projections:
ways to lose a dimension

- Another common one is **parallel projection**



Projections:
ways to lose a dimension

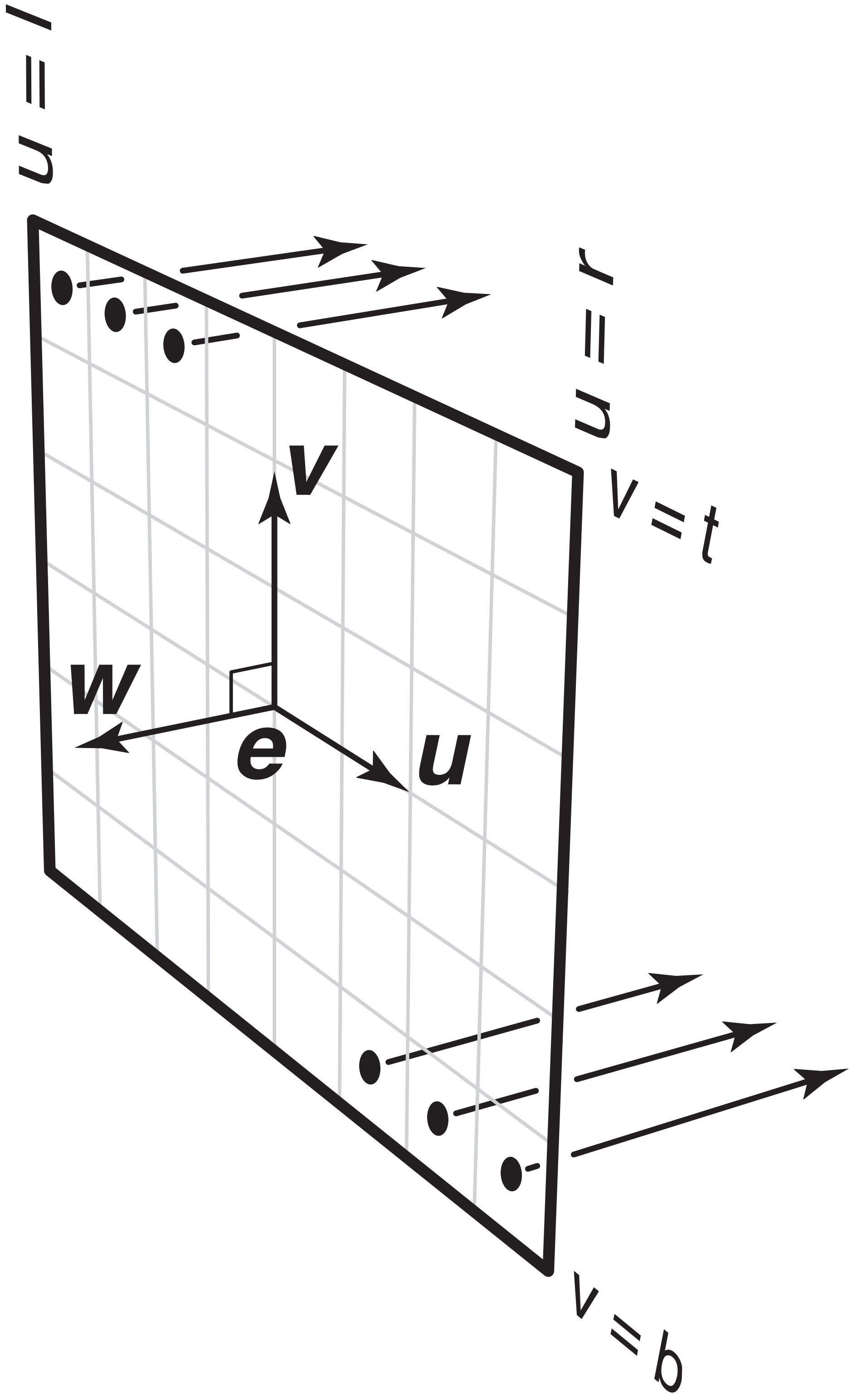
- Another common one is **parallel projection**



Projections: ways to lose a dimension

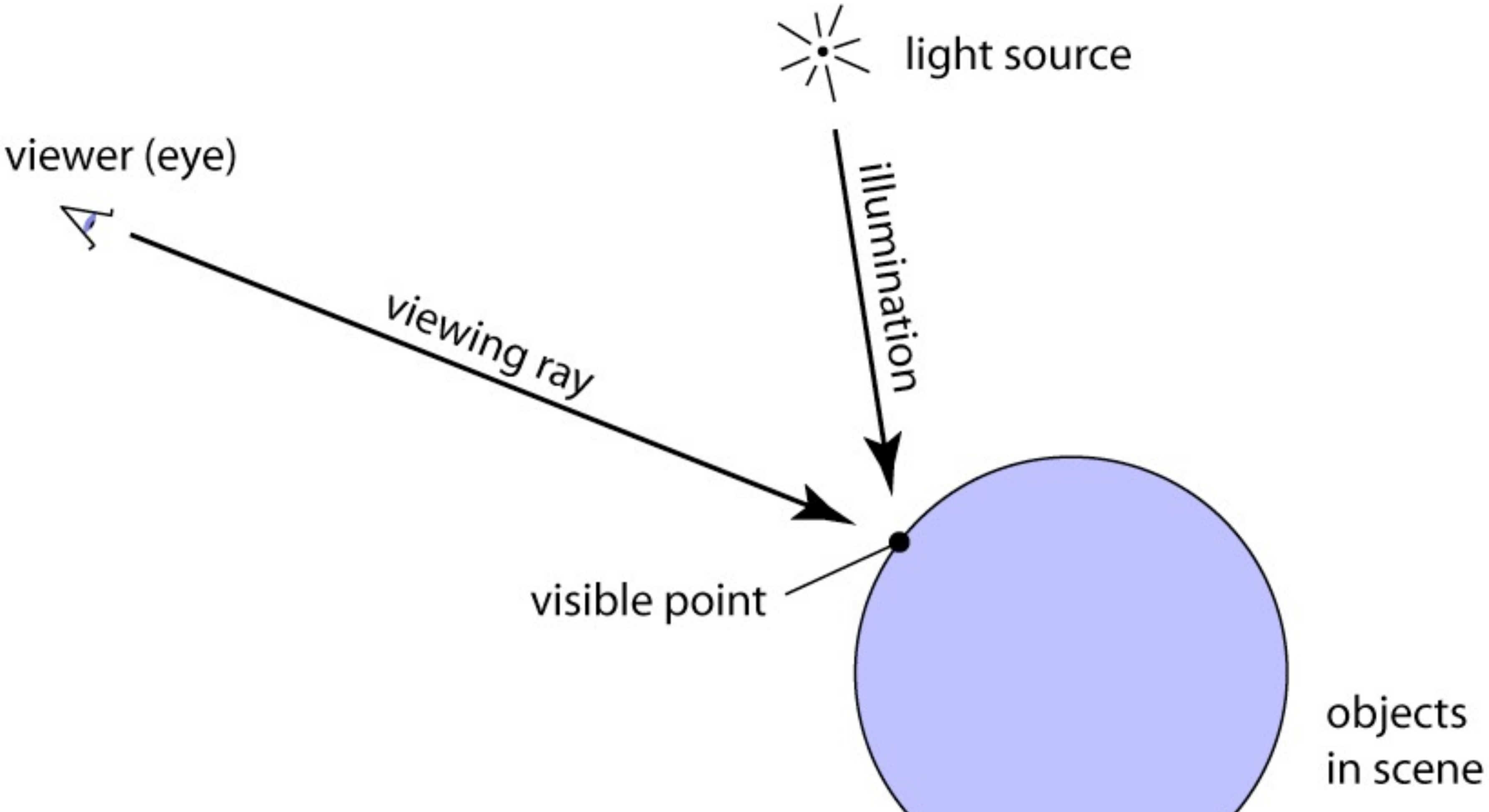
- Another common one is **parallel projection**

- Key property of parallel projections: all **viewing rays** are parallel



Ray Tracing: Pseudocode

```
for each pixel:
    generate a viewing ray for the pixel
    find the closest object it intersects
    determine the color of the object
```

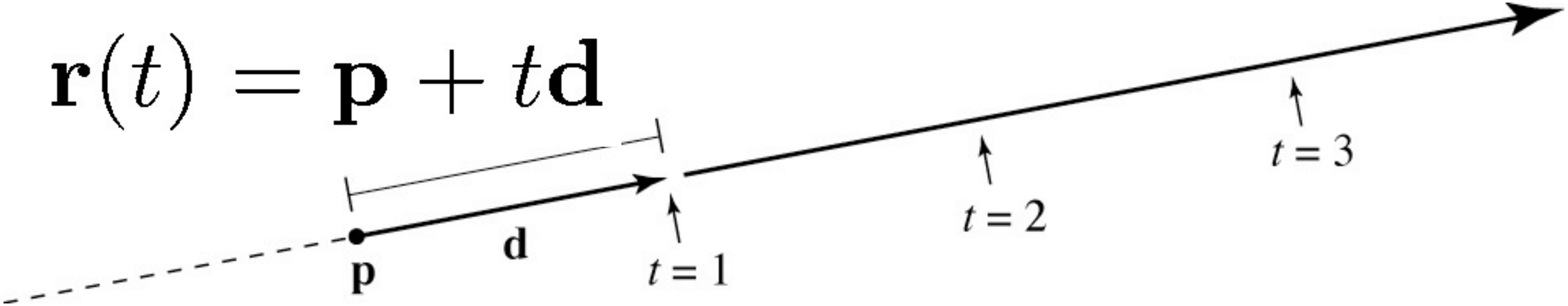


A ray is half a line.

We'll describe rays using:

- An *origin* (**p**) where the ray begins
- A *direction* (**d**) in which the ray goes

$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$

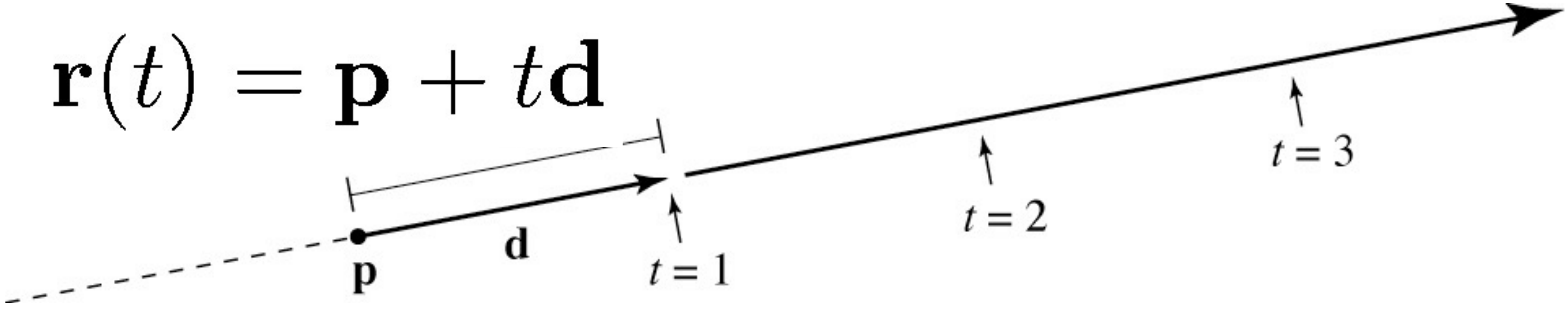


A ray is half a line.

We'll describe rays using:

- An *origin* (**p**) where the ray begins
- A *direction* (**d**) in which the ray goes

$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$



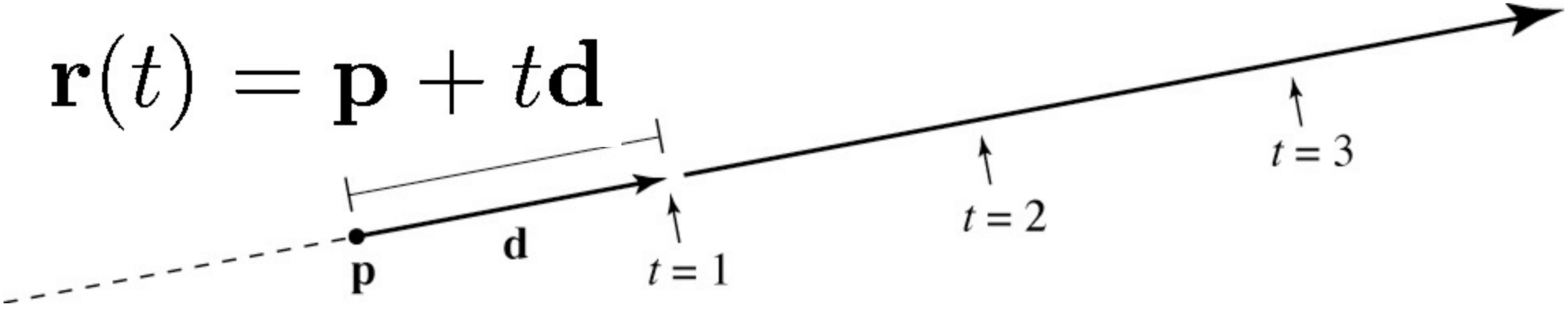
- This is a *parametric equation*: it **generates** points on the line

A ray is half a line.

We'll describe rays using:

- An *origin* (**p**) where the ray begins
- A *direction* (**d**) in which the ray goes

$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$

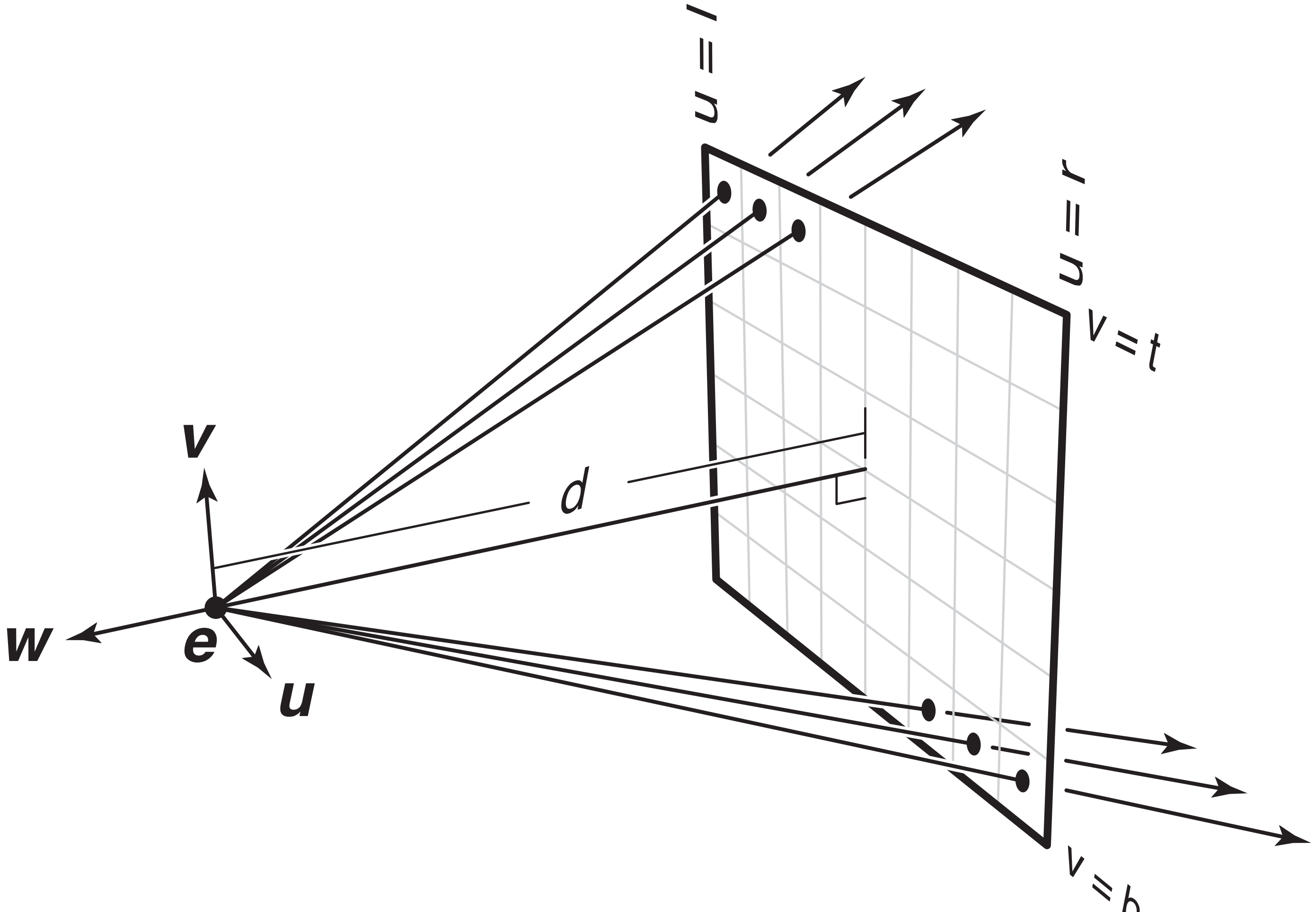


- This is a *parametric equation*: it **generates** points on the line
- The set of points with *t > 0* gives all points on the ray

Viewing Rays

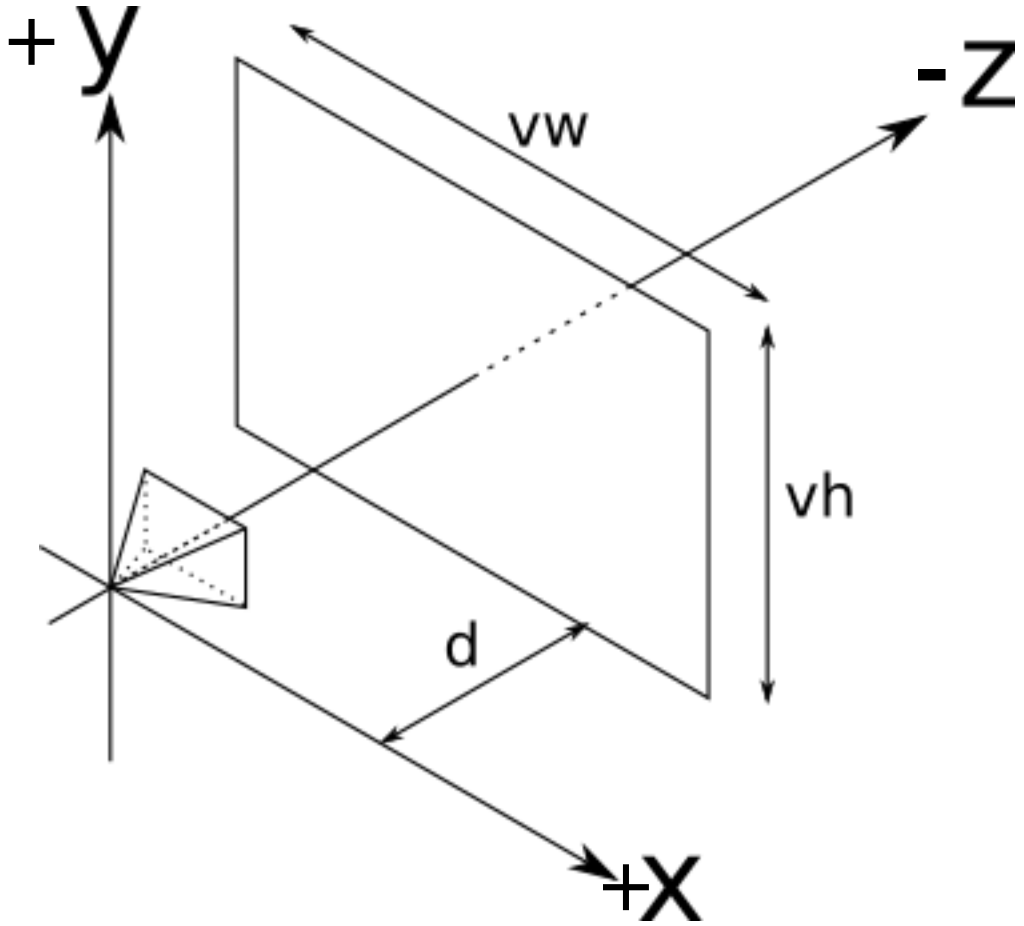
are determined by the **position** and **orientation** of the camera

- For perspective projection, viewing rays originate at the **eye**.
- The direction varies depending on the pixel.



Let's start with a simple camera

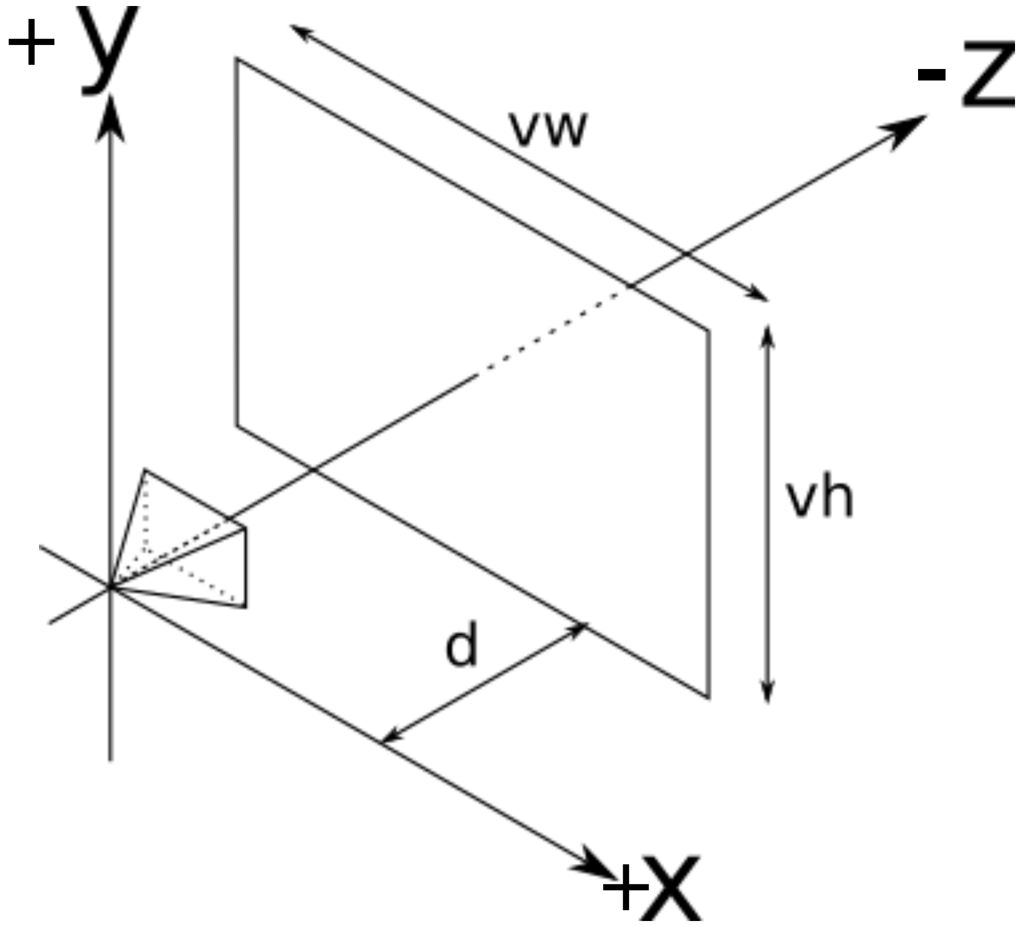
- Eye is at the origin (0, 0, 0)
- Looking down the **negative** z axis
- Viewport is aligned with the xy plane
- vh = vw = 1
- d = 1



What is the 3D viewing ray for pixel (i, j)?

Viewing rays for the canonical camera

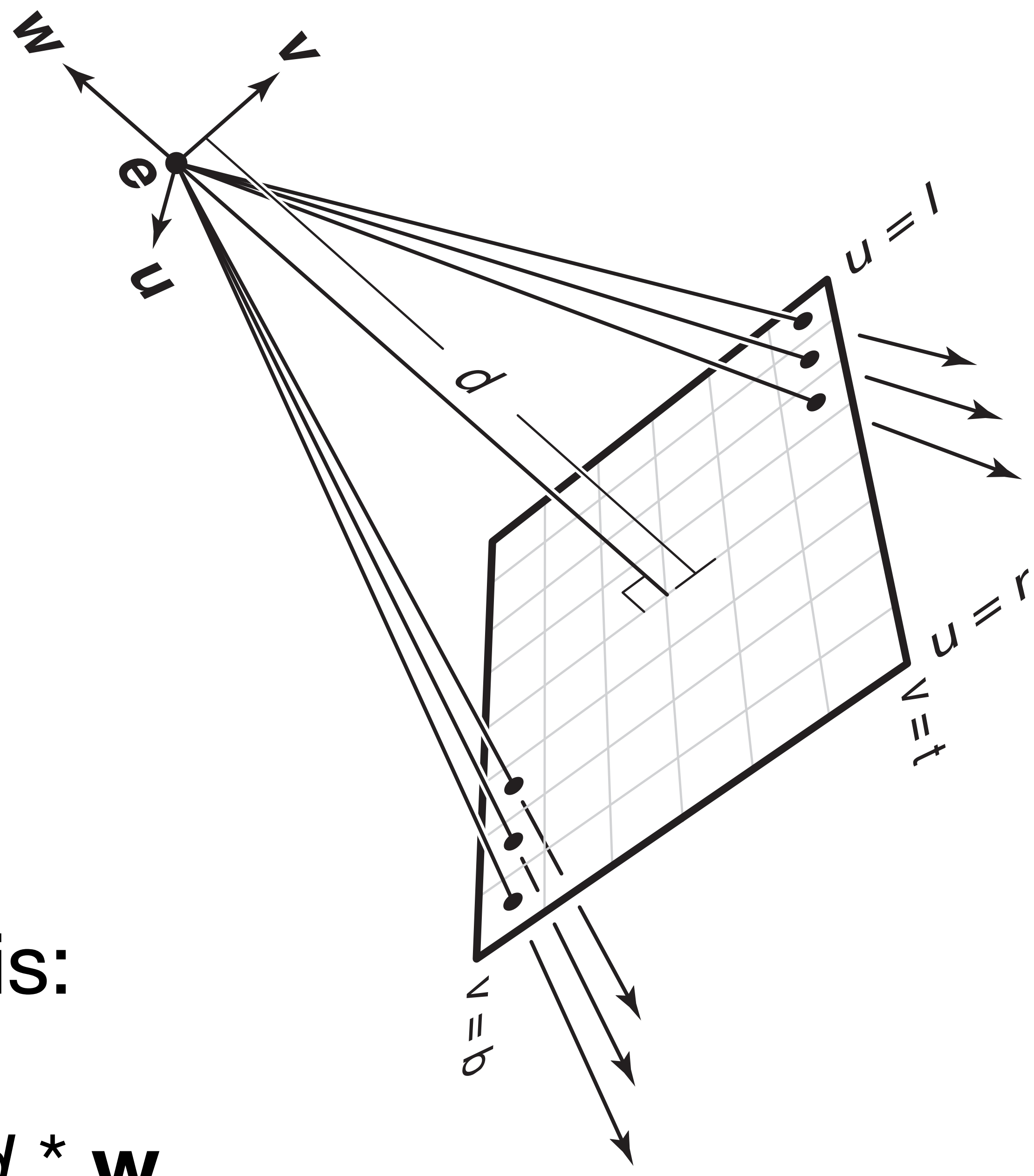
- u = (j - 0.5) / W - 0.5
- v = -((i - 0.5) / H - 0.5)
- The viewing ray is:
    - Origin: (0,0,0)
    - Direction: (u, v, -d)



What if I want to put the camera somewhere else?

The camera's pose is defined by a **coordinate system:**

- **u** points right from the eye
- **v** points up from the eye
- **w** points back from the eye



1. Turn (i,j) into *u, v* as before
2. Viewing ray in (x, y, z) world is:
   origin = eye
   direction = *u* * **u** + *v* * **v** + -*d* * **w**