Synthesizing Images

Lecture 2

Christian A. Pagot

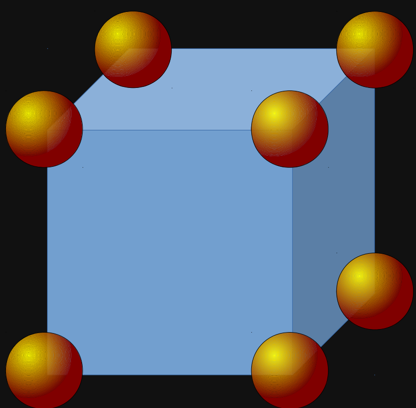
Universidade Federal da Paraíba
Centro de Informática

Rendering

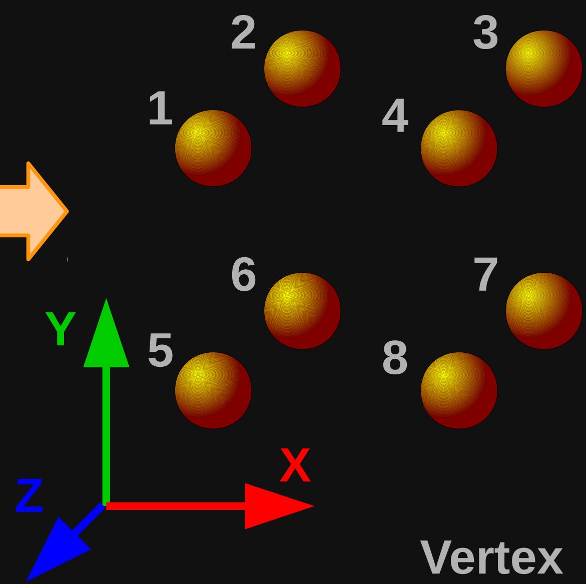
- There are several rendering techniques around.
- The two more prominent are:
  - Rasterization.
  - Ray Tracing.

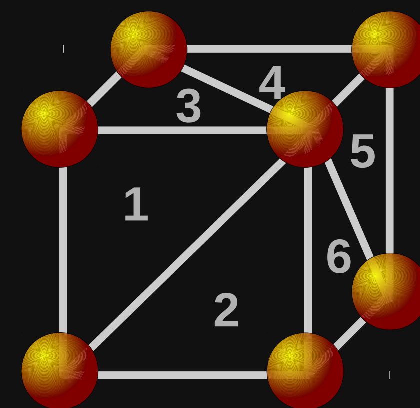
Rasterization Pipeline

Modeling



Object





Vertex list

1	5	5	5
2	5	5	1
3	9	5	1
4	9	5	5

⋮

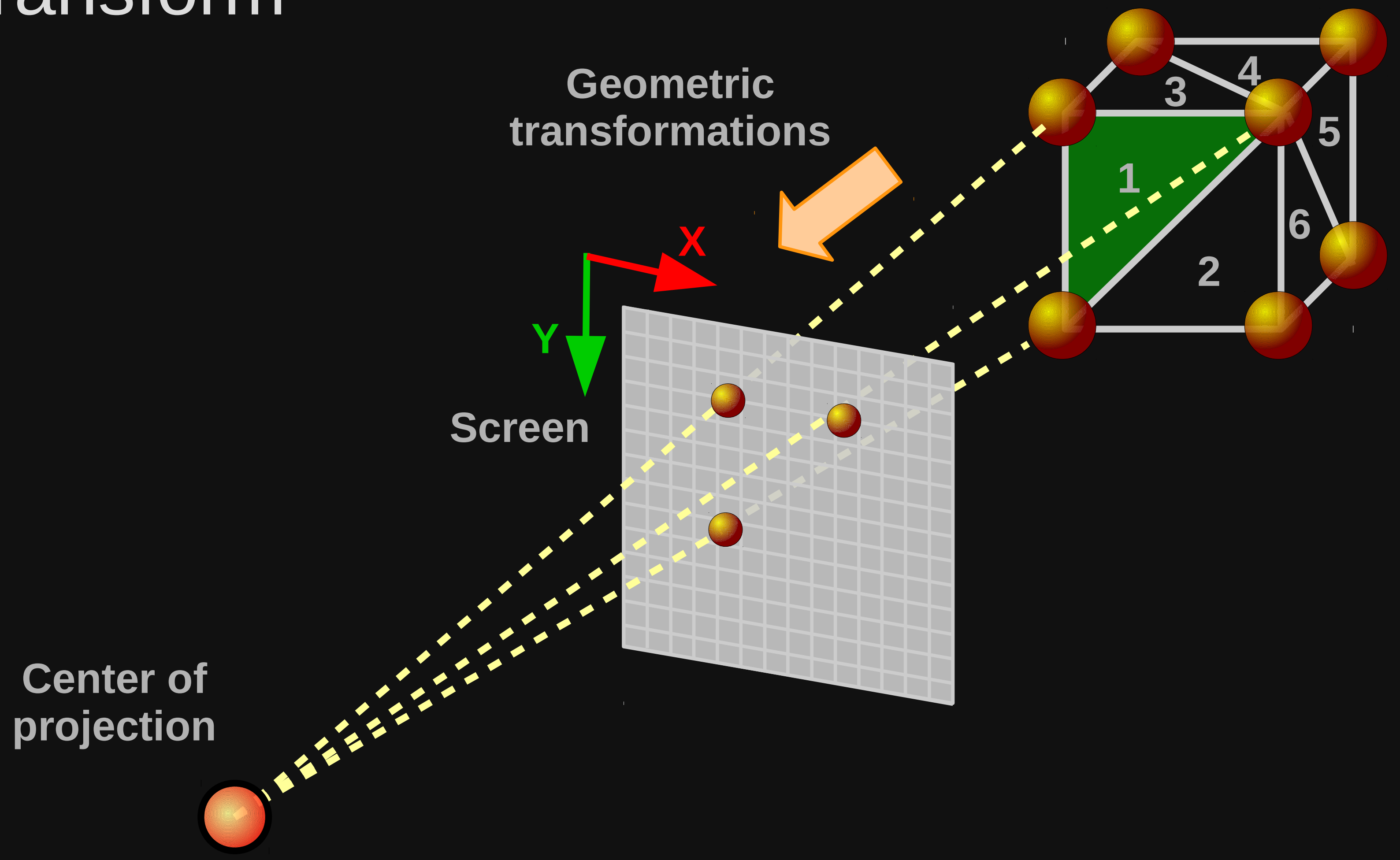
Triangle list

1	1	4	5
2	5	4	8
3	2	1	4
4	3	2	4

⋮

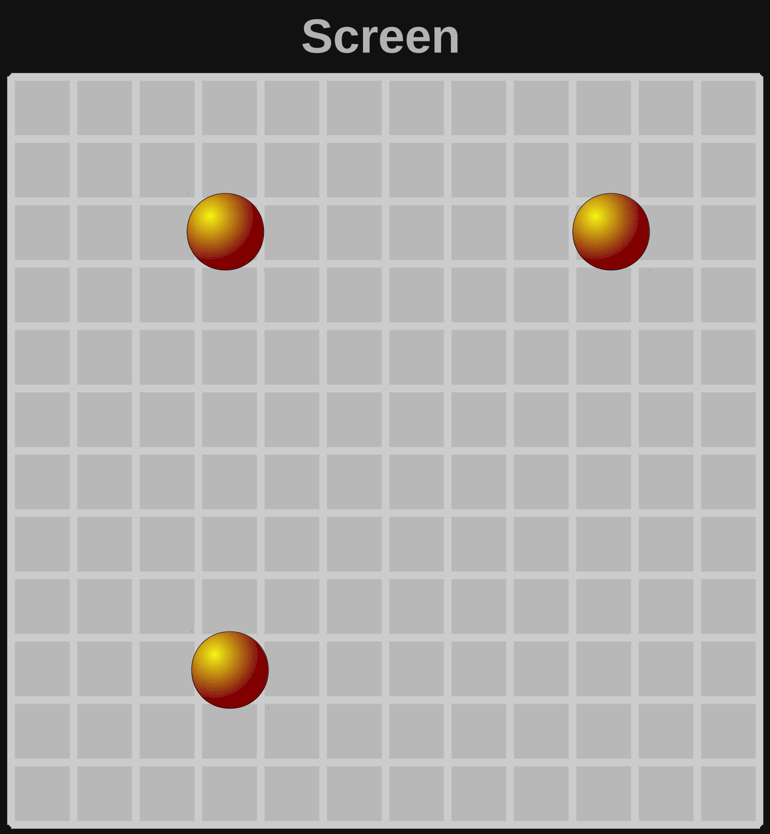
Rasterization Pipeline

Transform



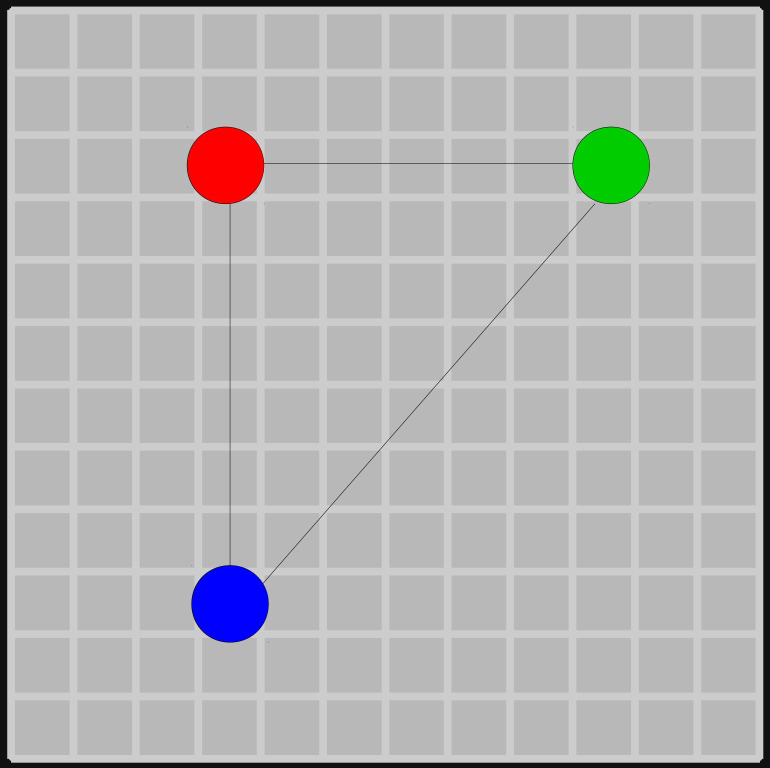
Rasterization Pipeline

Lighting



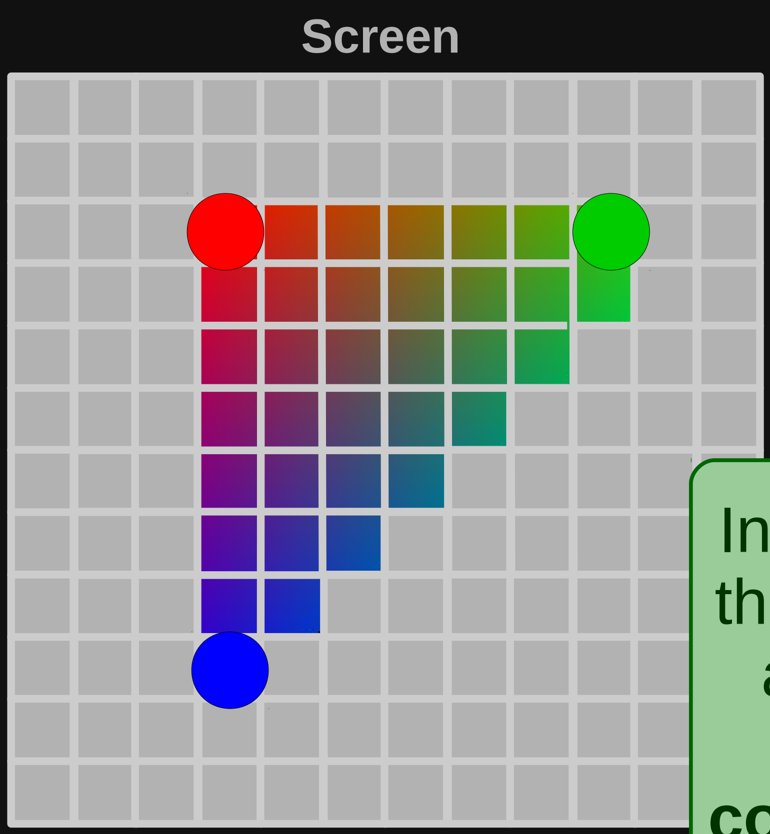
Rasterization Pipeline

Lighting



Rasterization Pipeline

Primitive rasterization



In ***per pixel shading*** the **vertex attributes** are **interpolated**, and **shading** is **computed** separately for **each fragment**!

Rasterization Pipeline

Final Result (including tricks to improve realism!)

Rasterization Pipeline

- Pros

  It is fast

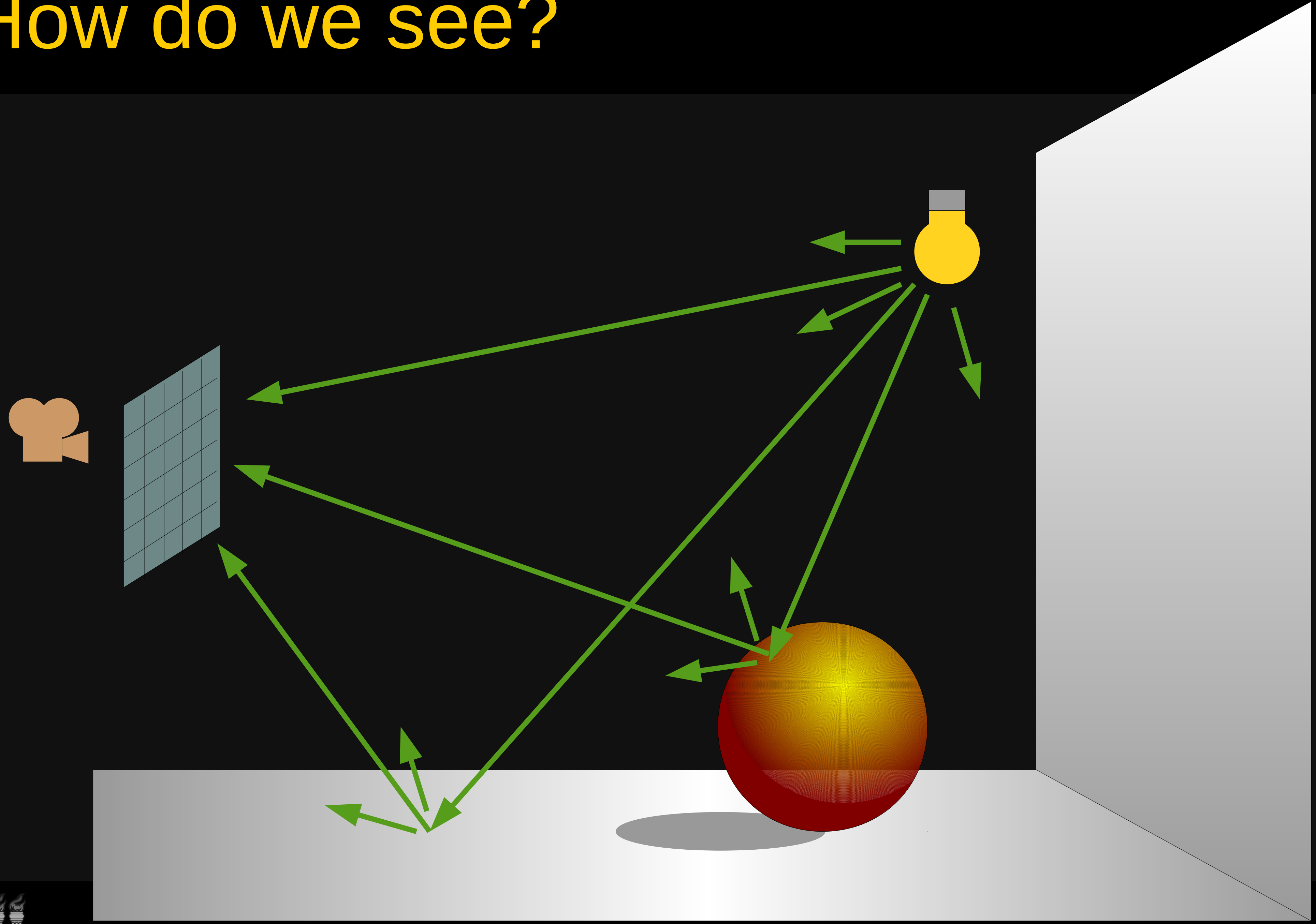
  - Primitives/fragments can be processed in parallel.
  - Fragment properties can be inferred through interpolation.

- Cons

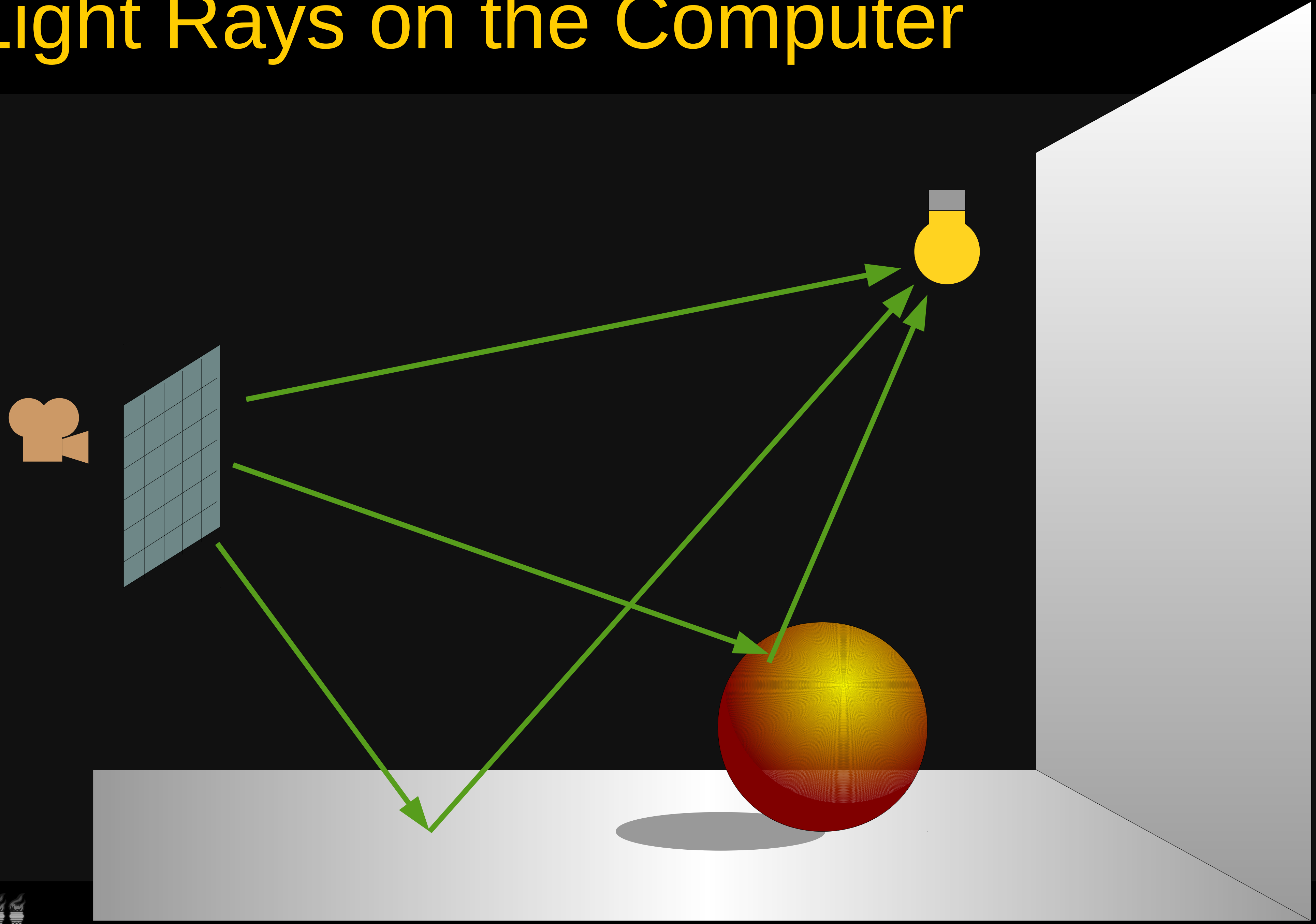
  It is not realistic

  - Does not take into account the interaction of luminous energy with the surfaces.
  - Almost every global illumination effect must be computed separately (shadows, refraction, reflection, translucency...).

How do we see?



Light Rays on the Computer

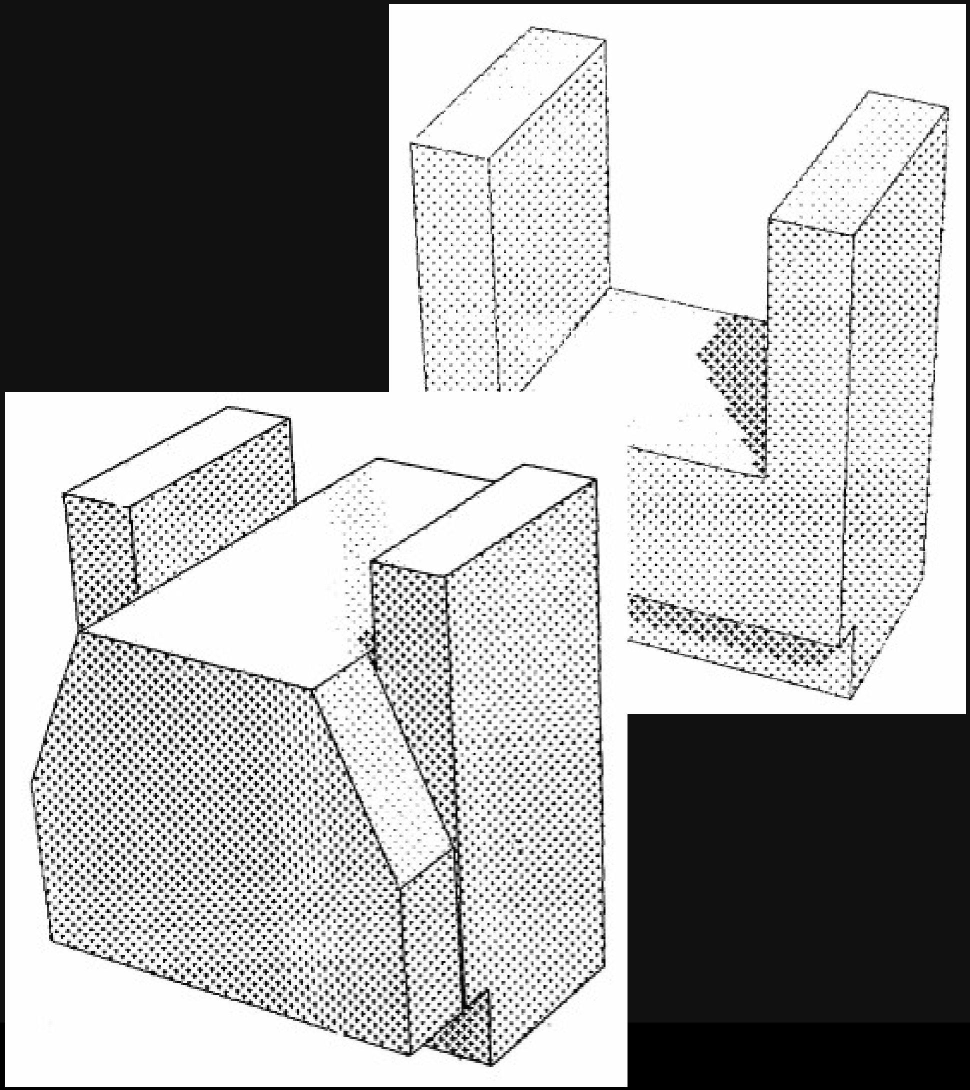


Ray Casting

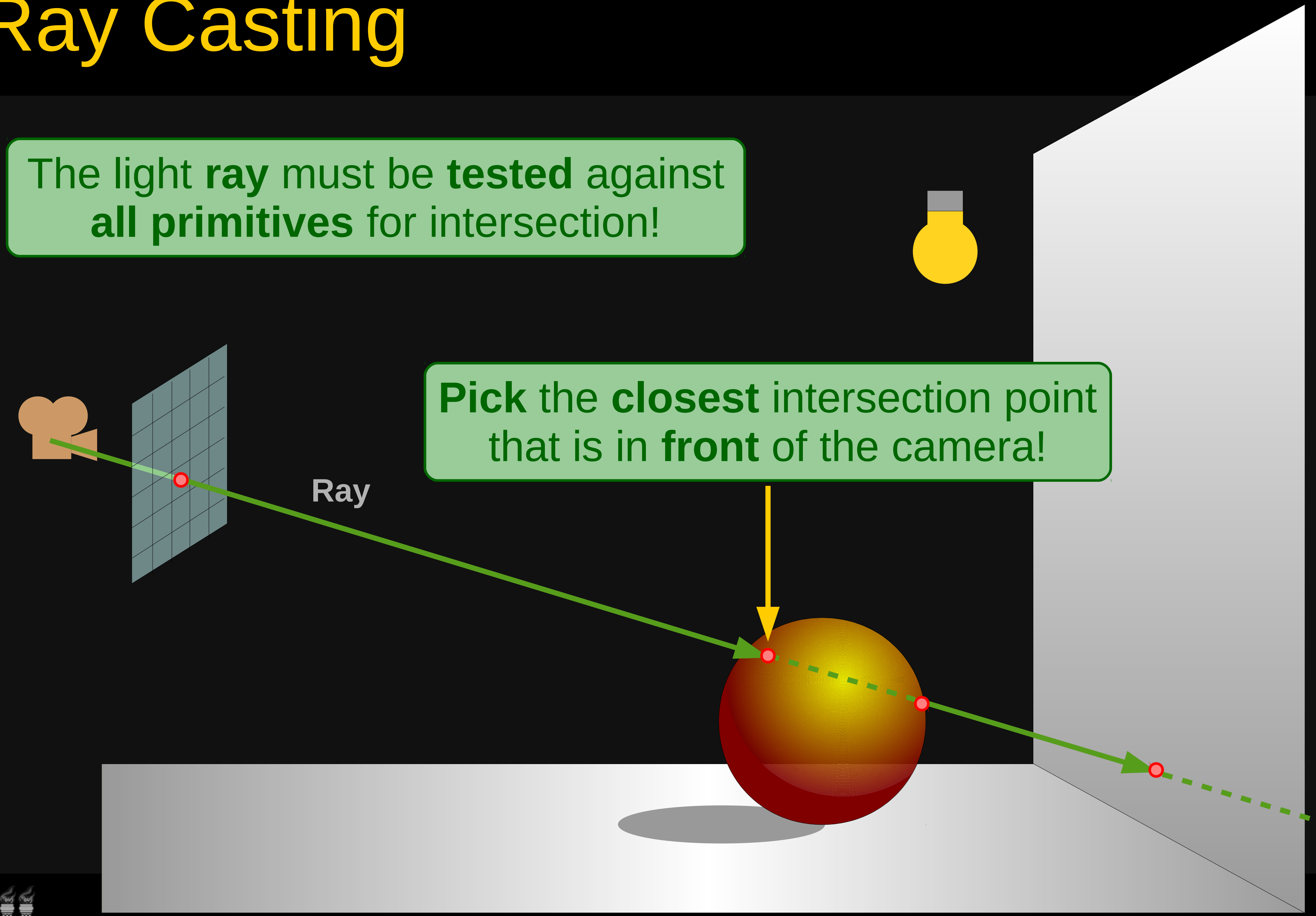
In 1968 Appel generated images using particle tracing:

Some techniques for shading machine renderings of solids.

A. Appel, 1968.



Ray Casting



The light **ray** must be **tested** against **all primitives** for intersection!
Ray
Pick the **closest** intersection point that is in **front** of the camera!

The Rendering Equation

In 1986 James Kajiya rewrote the problem of light transport as an integral, which he called **The Rendering Equation**:

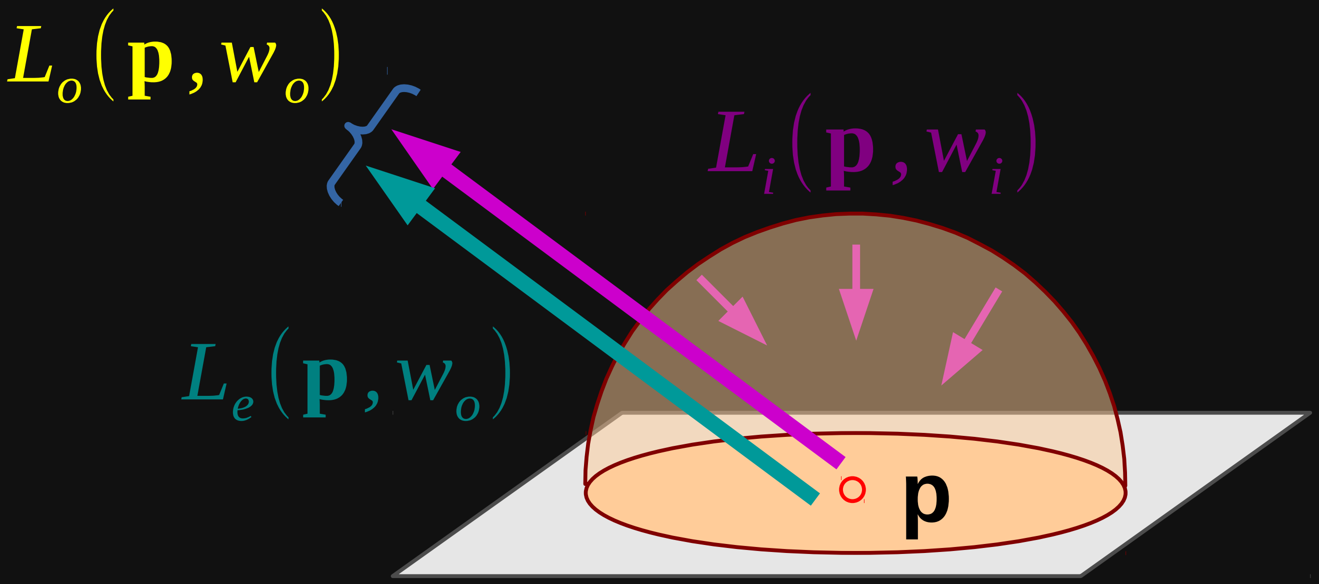
$$L_o(\mathbf{p}, w_o) = L_e(\mathbf{p}, w_o) + \int_{\Omega} f_r(\mathbf{p}, w_o, w_i)\, L_i(\mathbf{p}, w_i) \cos\theta_i \, dw_i$$

The Rendering Equation

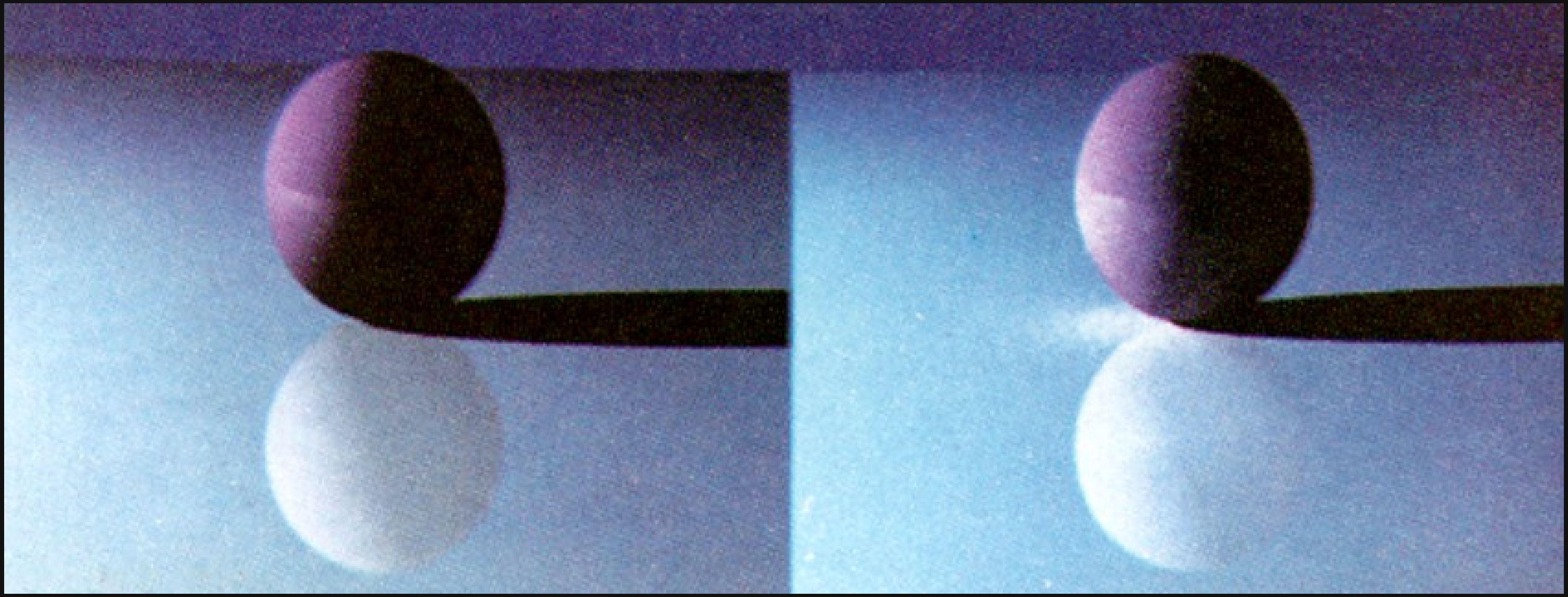
J. Kajiya, 1986.

The Rendering Equation

$$L_o(\mathbf{p}, w_o) = L_e(\mathbf{p}, w_o) + \int_\Omega f_r(\mathbf{p}, w_o, w_i)\, L_i(\mathbf{p}, w_i) \cos\theta_i \, dw_i$$

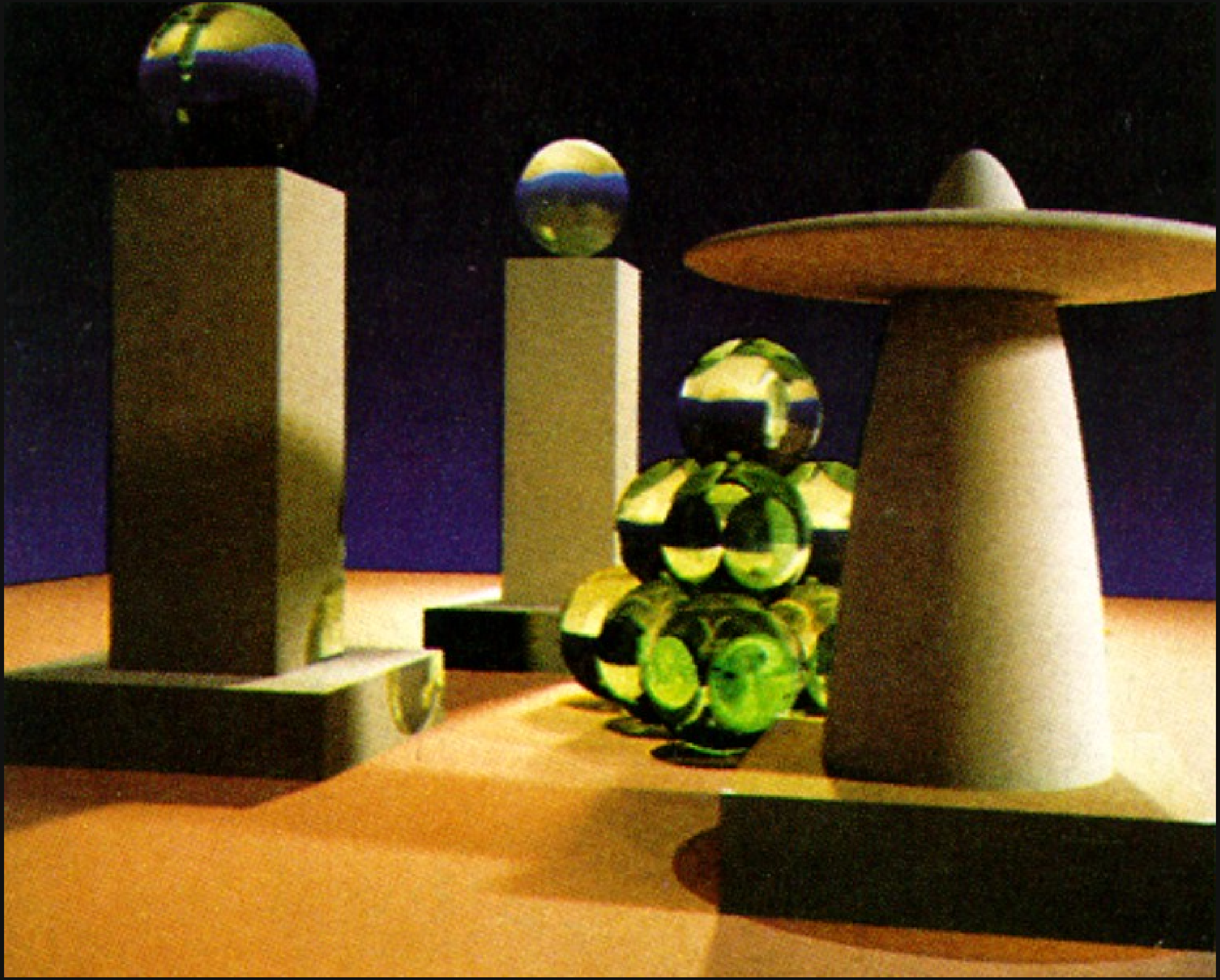


The Rendering Equation



Ray Tracing Path Tracing

The Rendering Equation



The Rendering Equation

Jaguar
Maxuell Render

The Rendering Equation

Geometric Transformations

Lecture 3

Christian A. Pagot

Universidade Federal da Paraíba
Centro de Informática

Transformation Types

- Scale
- Rotation
- Translation

Transformations in 3D Space

- **3D transformations** can also be represented through **matrices**.
- As in the 2D case, **3D transformations** are expressed with the help of **HC** to allow for the representation of translation in matrix form.
- 3D vectors in **Euclidean** space are represented through **4-element vectors in HC**:

Example: $(x, y, z) \rightarrow (xw, yw, zw, w)$

3D Scale

The scale can be computed along X, Y and Z axis

$$x' = x \cdot s_x$$
$$y' = y \cdot s_y$$
$$z' = z \cdot s_z$$

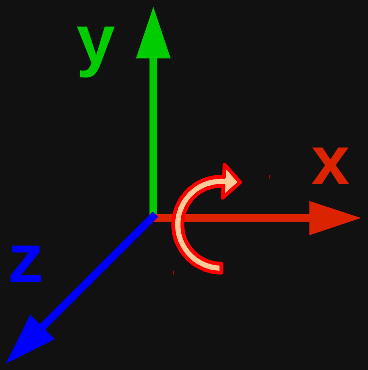
Matrix Form in HC

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D Rotation

- Rotation is computed always ...
  - … about an axis.
- There are three possibilities for a rotation in 3D space:
  - About X, Y, or Z axis.

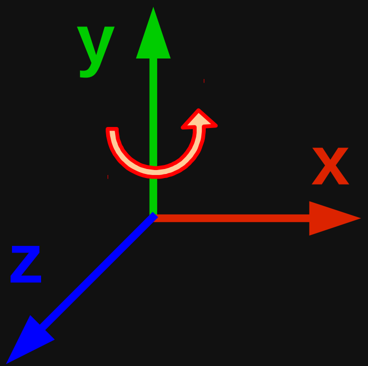
3D Rotation about X



Matrix Form in HC

$$
\begin{aligned}
x' &= x \\
y' &= y\cos(\theta) - z\sin(\theta) \\
z' &= y\sin(\theta) + z\cos(\theta)
\end{aligned}
\quad \Rightarrow \quad
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
$$

3D Rotation about Y

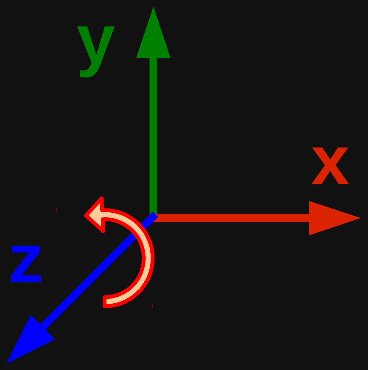


Matrix Form in HC

$$x' = x\cos(\theta) + z\sin(\theta)$$
$$y' = y$$
$$z' = -x\sin(\theta) + z\cos(\theta)$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D Rotation about Z



$$x' = x\cos(\theta) - y\sin(\theta)$$
$$y' = x\sin(\theta) + y\cos(\theta)$$
$$z' = z$$

Matrix Form in HC

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D Translation

An additional dimension transforms the 3D translation in a 4D shear:

$$\begin{aligned} x' &= x + d_x \\ y' &= y + d_y \\ z' &= z + d_z \end{aligned} \Rightarrow$$

Matrix Form in HC

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Ray Casting

Lecture 4

Christian A. Pagot

Universidade Federal da Paraíba
Centro de Informática

Ray Casting

Algorithm for a “shadowless” ray casting

```
for each image pixel
    shoot a ray from the camera position through the pixel center

    for each primitive
        test the intersection of the ray with the primitive

    if the ray intersects one or more primitives
        return the primitive color for the closest intersection point
    else
        return the background color
```

Ray Structure

- Mathematical Model

  A very common ray model is based on the 3D parametric line model

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

  where r is the **ray line segment**, o is the **ray origin**, d is the **normalized direction** and *t* is the parameter that identifies a **particular position** along the ray.

- Useful Properties
  - One can determine the **distance of point along the ray** (*t*).
  - Negative *t* indicates a **point "behind"** the origin.

Camera

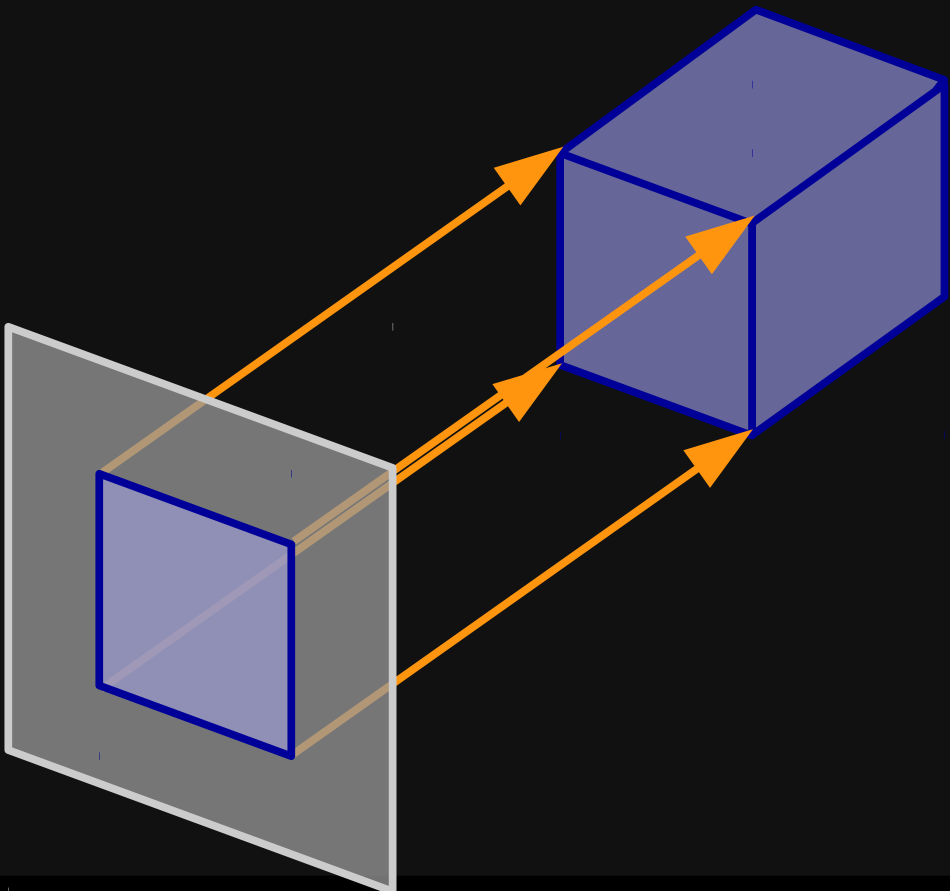
There are several possibilities for the camera model:

- Orthographic.
- Perspective (most common are *pinhole* or *thin lens*).
- Physically−based camera model.
- ...

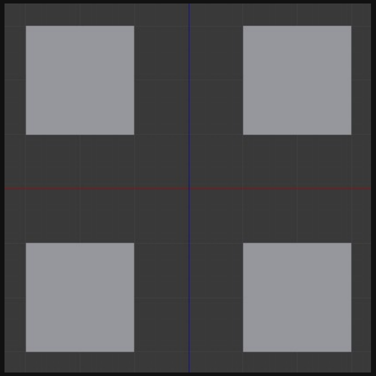
Orthographic Camera

- **Parallel rays.**
- **No** perspective **distortion.**

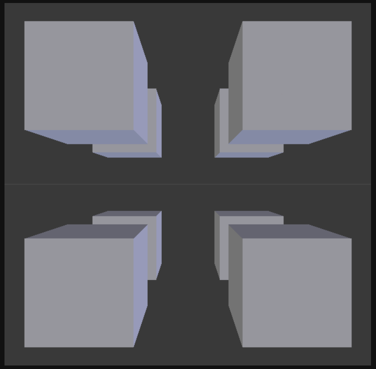
Projection plane



Orthographic projection

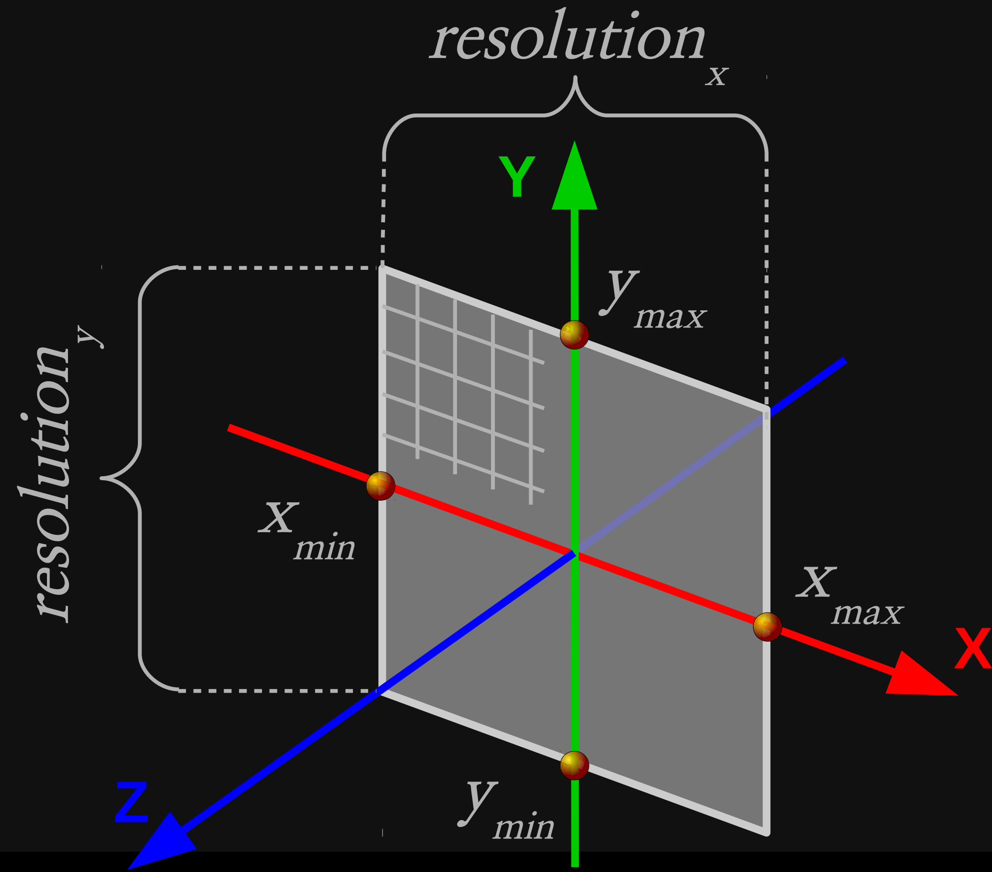


Perspective projection



Setting Up the Orthographic Camera

Suppose an orthographic camera sitting at the origin of the right-handed world space, pointing into -Z direction:

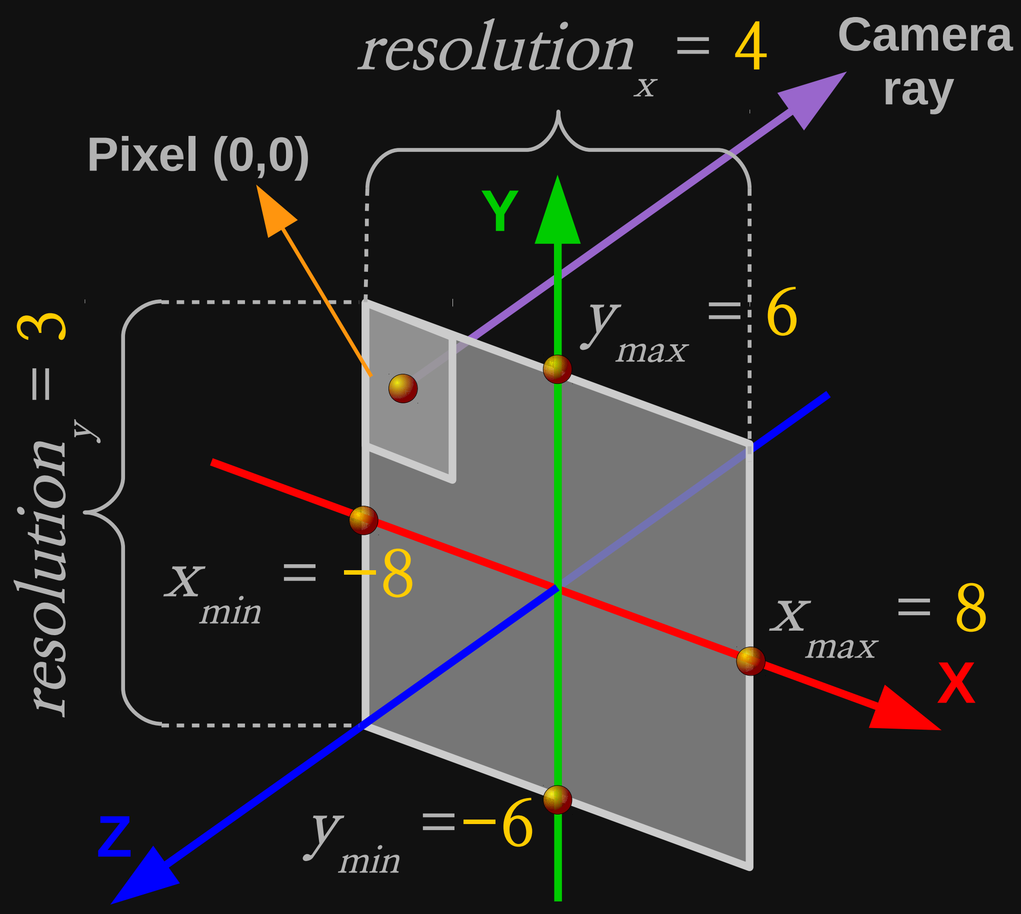


Which are **extents** of the camera **projection plane**?

Which is the **image resolution**?

How do we **generate** the **camera rays**?

Computing the Camera Rays



Computing the ray origin

$$width = x_{max} - x_{min}$$

$$height = y_{max} - y_{min}$$

$$\text{ray origin}_x = \frac{x_{pixel} + 0.5}{resolution_x} width + x_{min}$$

$$\text{ray origin}_y = \frac{y_{pixel} + 0.5}{resolution_y} height + y_{min}$$

$$\text{ray origin}_z = 0$$

Ray for pixel (0,0):

$$\text{ray origin} = (-6, 4, 0)$$

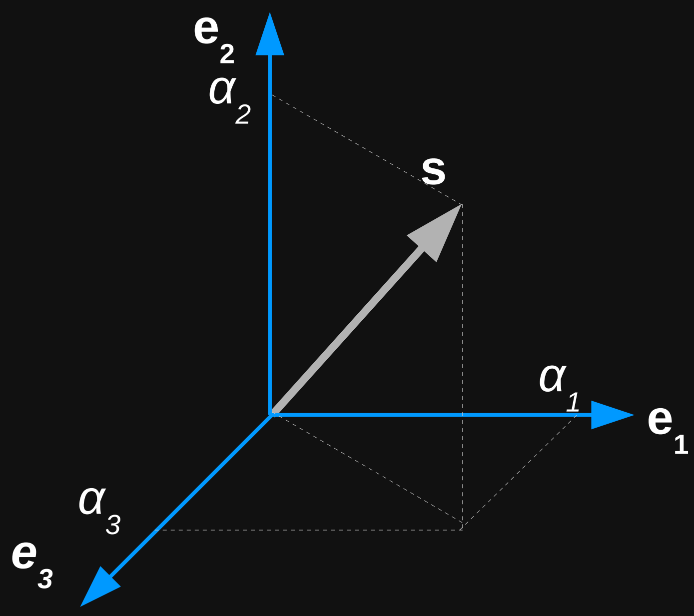
$$\text{ray direction} = (0, 0, -1)$$

Ray/Primitive Intersections

- Common primitives found in ray casters/tracers include:
  - Sphere.
  - Triangle.
  - Infinite plane.
  - Box.
  - Quadrics.
  - Etc.

Coordinate System Change

Consider an arbitrary vector **w** described in a given orthonormal basis:



$$\mathbf{s} = [\alpha_1 \mathbf{e_1} + \alpha_2 \mathbf{e_2} + \alpha_3 \mathbf{e_3}]$$

$$\mathbf{a} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}$$

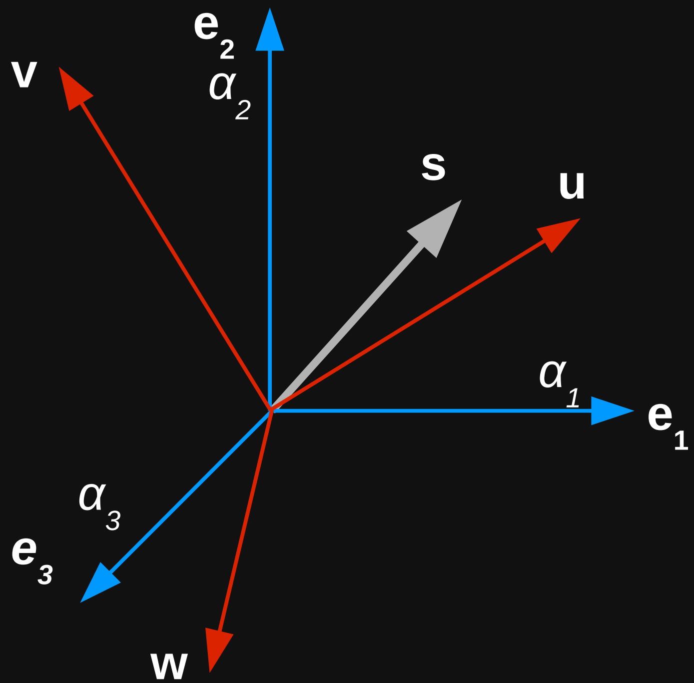
$$\mathbf{s} = [\mathbf{e_1} \quad \mathbf{e_2} \quad \mathbf{e_3}]\mathbf{a}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\boxed{\mathbf{s} = \mathbf{a}}$$

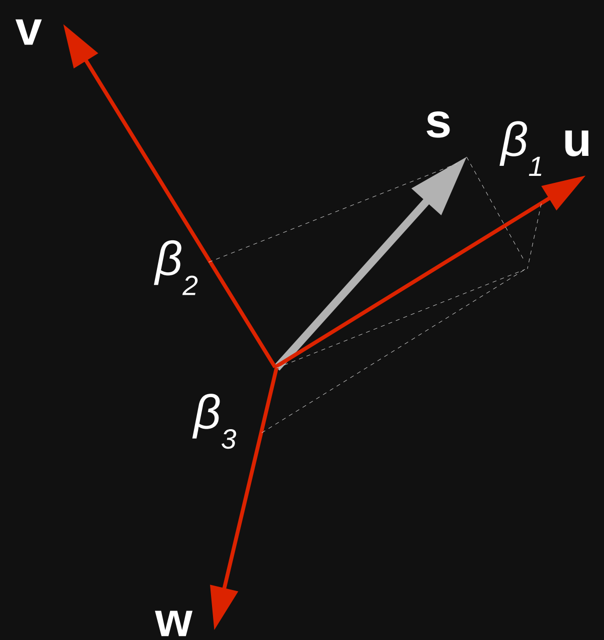
Coordinate System Change

Consider an arbitrary vector **w** described in a given orthonormal basis:



Coordinate System Change

Consider an arbitrary vector **w** described in a given orthonormal basis:



$$\mathbf{s} = \left[\beta_1 \mathbf{u} + \beta_2 \mathbf{v} + \beta_3 \mathbf{w}\right]$$

$$\mathbf{b} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}$$

$$\mathbf{s} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} \end{bmatrix} \mathbf{b} \qquad \mathbf{B} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} \end{bmatrix}$$

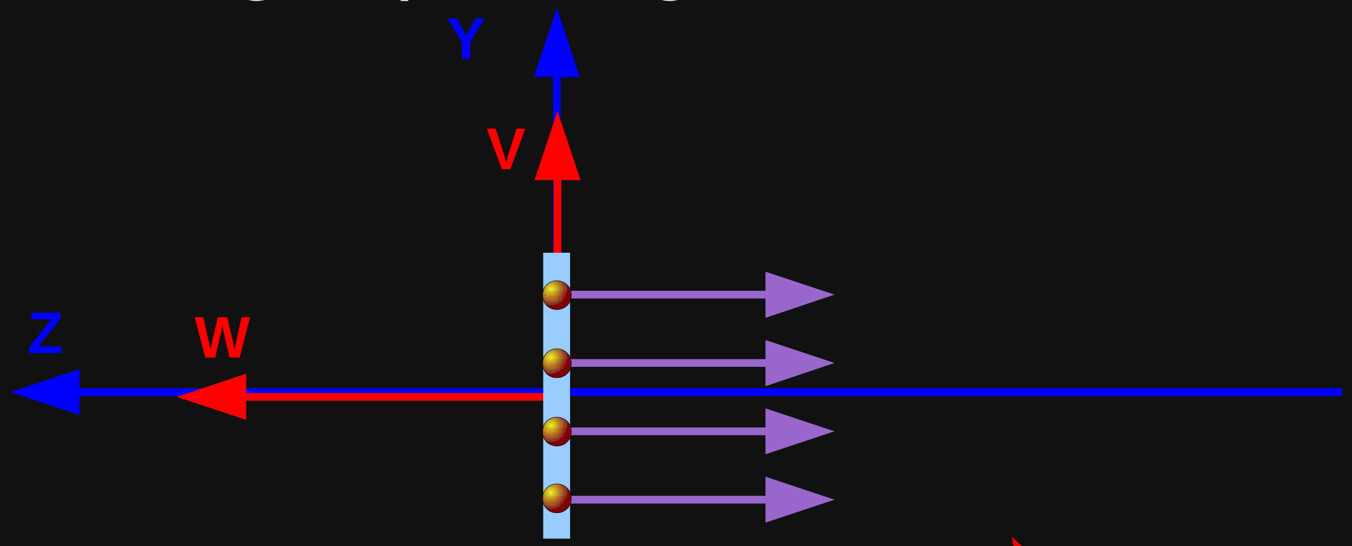
$$\mathbf{s} = \mathbf{B}\,\mathbf{b}$$

$$\mathbf{s} = \mathbf{a}$$

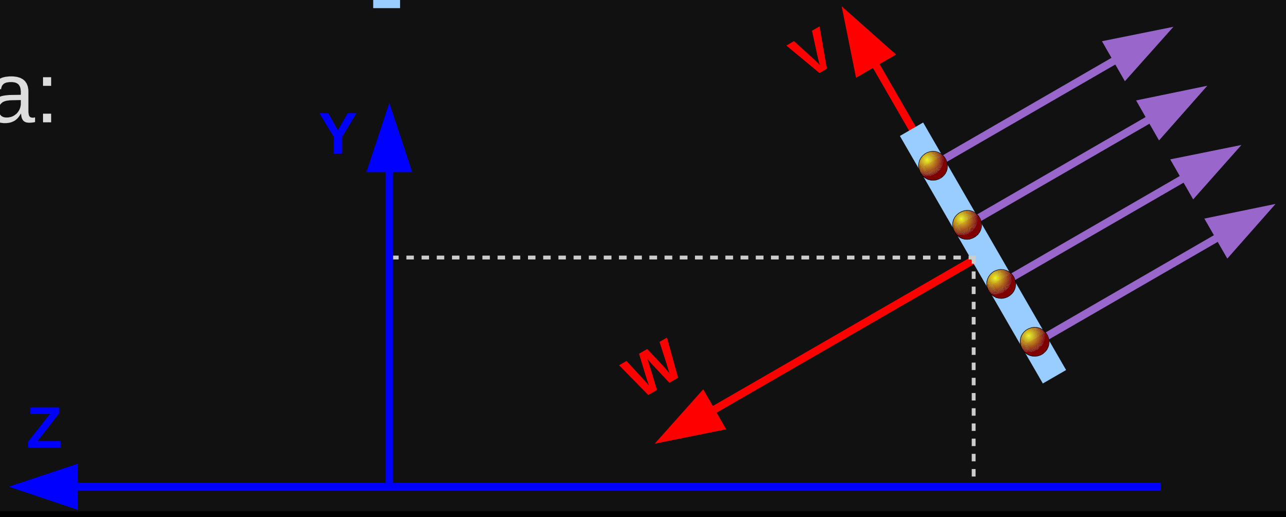
$$\boxed{\mathbf{a} = \mathbf{B}\mathbf{b}}$$

Arbitrary Orthographic Camera

- Camera at the origin, pointing at -Z:



- Arbitrary camera:



Arbitrary Orthographic Camera

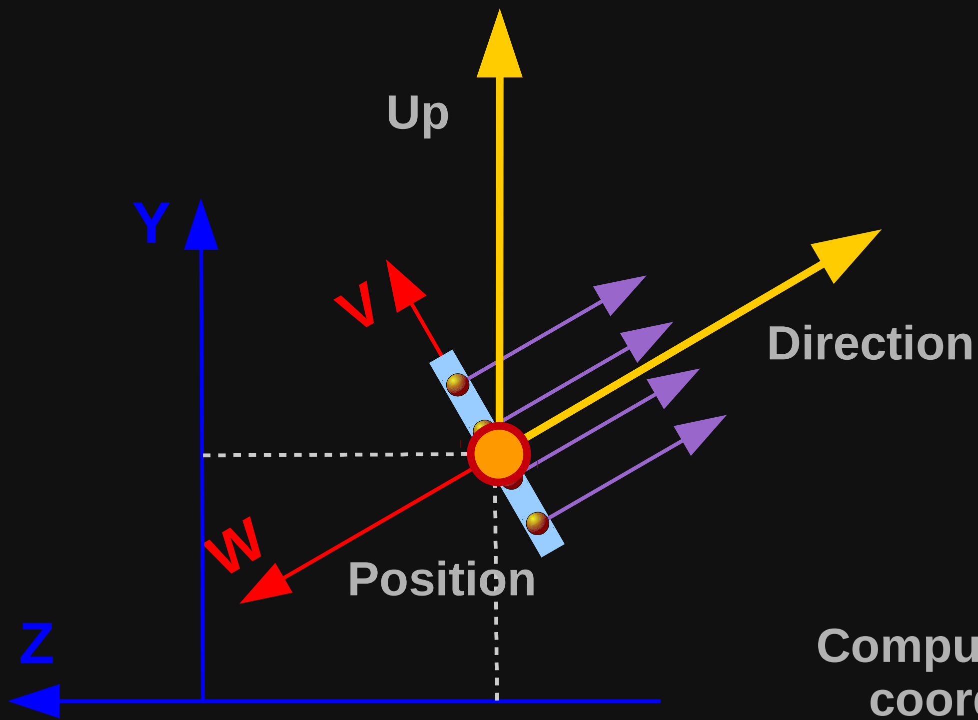
There are two alternatives:

- Transform the objects from the world to the local camera space.
- Transform the rays from the camera to the world space.

We will keep with the second option: transform the rays from the camera to world space!

Setting Up an Arbitrary Ortho. Camera

Constructing the camera coordinate system



Camera position: $\mathbf{p} = (x_{\mathbf{p}}, y_{\mathbf{p}}, z_{\mathbf{p}})$

Camera direction: $\mathbf{d} = (x_{\mathbf{d}}, y_{\mathbf{d}}, z_{\mathbf{d}})$

Up vector: $\mathbf{k} = (x_{\mathbf{k}}, y_{\mathbf{k}}, z_{\mathbf{k}})$

$$\mathbf{w} = \frac{\mathbf{d}}{|\mathbf{d}|}$$

Computing the camera's coord. System base

$$\mathbf{u} = \frac{\mathbf{k} \times \mathbf{w}}{|\mathbf{k} \times \mathbf{w}|}$$

$$\mathbf{v} = \frac{\mathbf{w} \times \mathbf{u}}{|\mathbf{w} \times \mathbf{u}|}$$

Setting Up an Arbitrary Ortho. Camera

- First, compute the origin and the direction of the rays in the local camera space:

$$\text{ray origin}_x = \frac{x_{pixel} + 0.5}{resolution_x} width + x_{min}$$

$$\text{ray origin}_y = \frac{y_{pixel} + 0.5}{resolution_y} height + y_{min}$$

$$\text{ray origin}_z = 0$$

$$\text{ray direction} = (0, 0, -1)$$

- Transform the ray origin and direction to the world space:

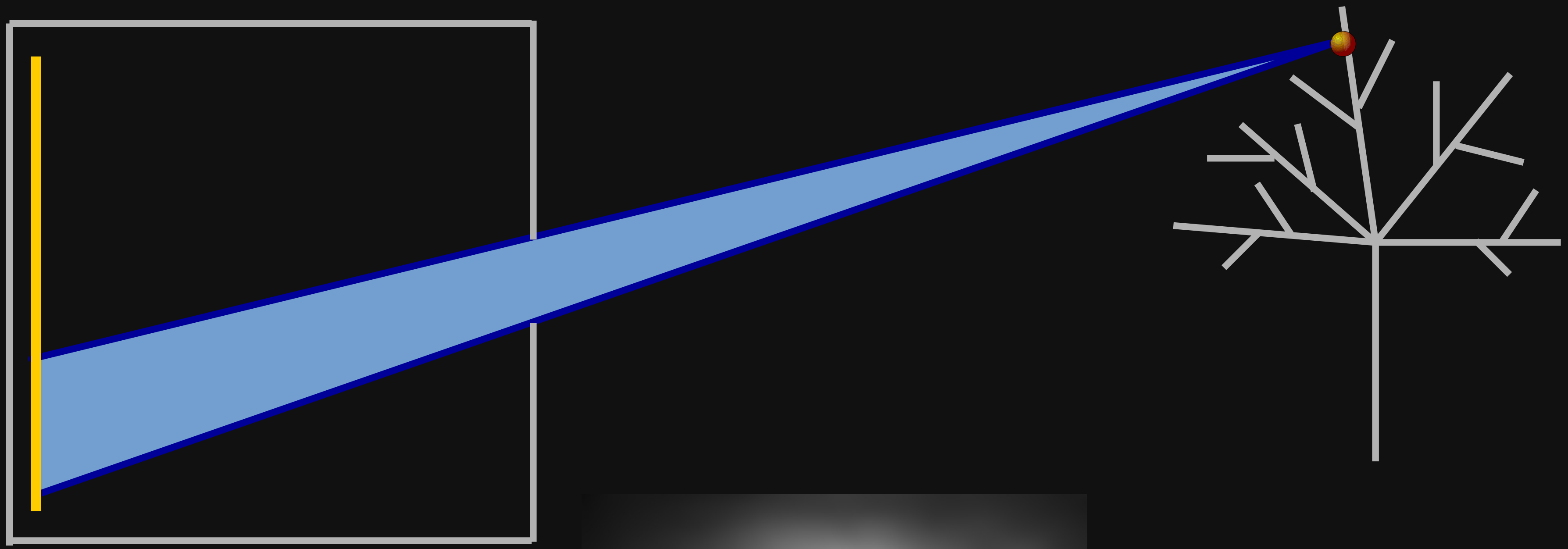
$$\text{ray origin}_{\text{world}} = \mathbf{B}\, \text{ray origin}_{\text{cam}} + \mathbf{p}$$

$$\text{ray direction}_{\text{world}} = \mathbf{B}\, \text{ray direction}_{\text{cam}}$$

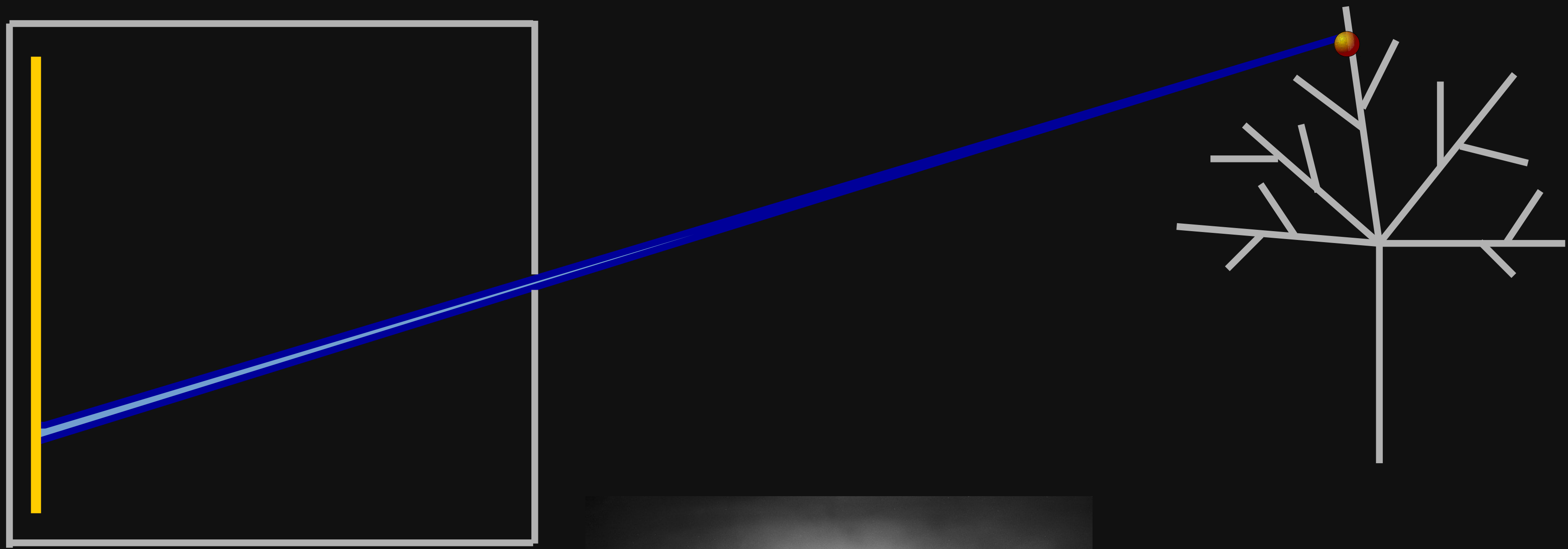
Perspective (pinhole) Camera

- Camera **rays** emanate from the camera **center of projection** and go through **each pixel**.
- There is **perspective distortion**.
- Everything is **in focus**.

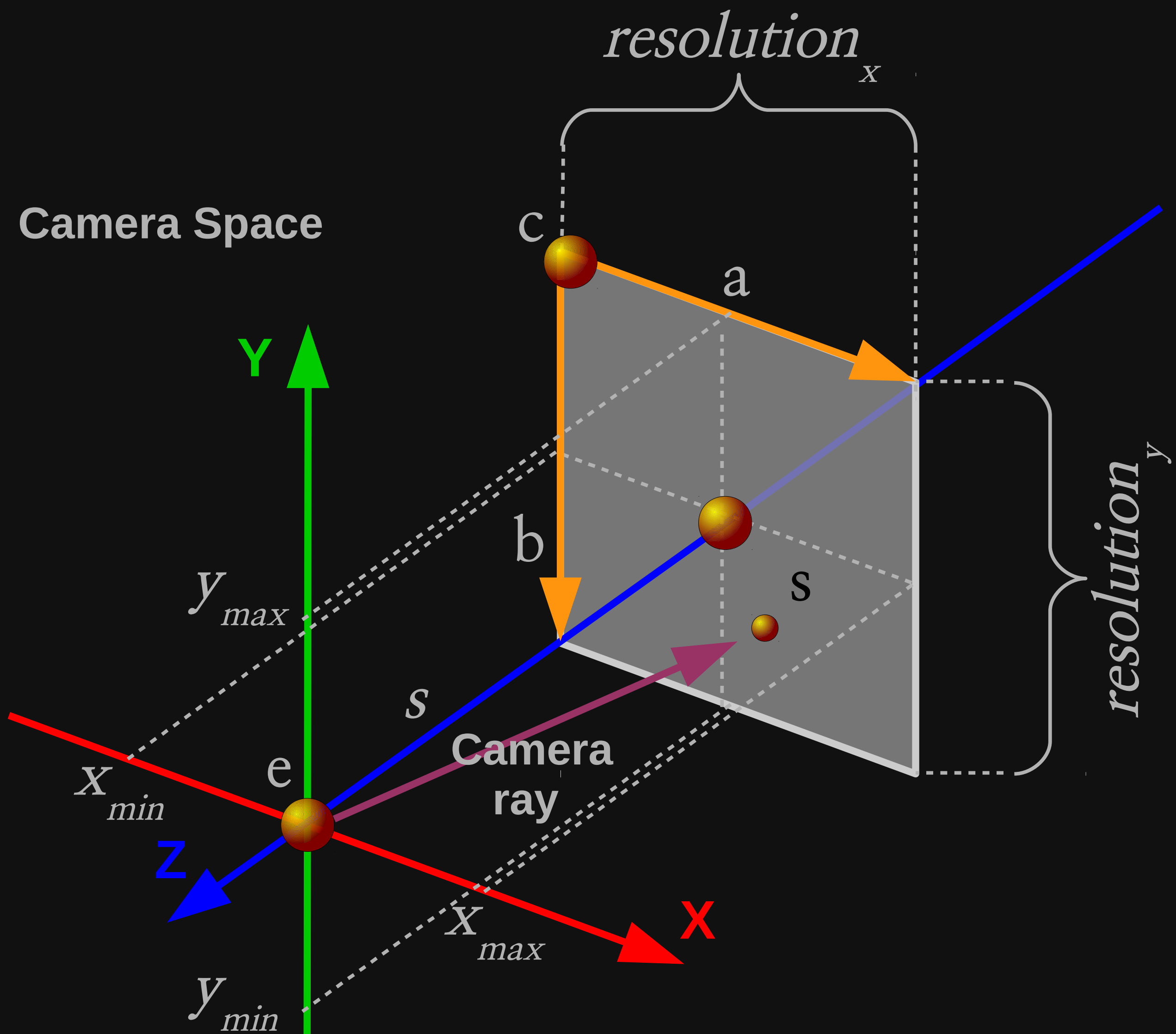
Perspective (pinhole) Camera



Perspective (pinhole) Camera



Pinhole Camera Setup



Camera vectors

$$\mathbf{a} = (x_{max} - x_{min})$$

$$\mathbf{b} = (y_{max} - y_{min})$$

$$\mathbf{c} = x_{min}\,\mathbf{x} + y_{min}\,\mathbf{y} - s\,\mathbf{z}$$

A point on screen

$$\mathbf{s} = \mathbf{c} + u\,\mathbf{a} + v\,\mathbf{b}$$

A ray from e to s

$$\mathbf{ray} = \mathbf{e} + t(\mathbf{s} - \mathbf{e})$$

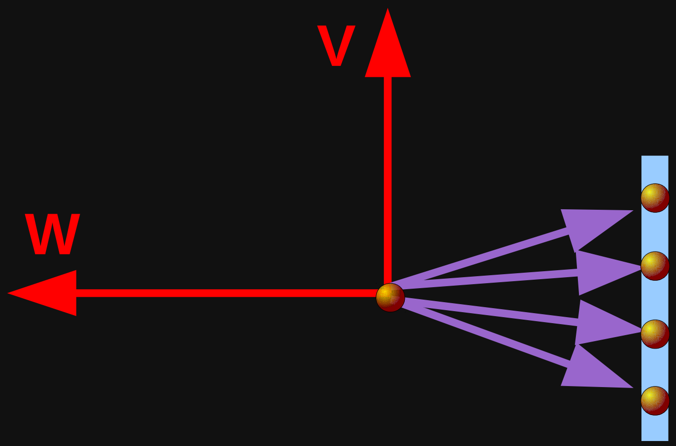
From a (x,y) pixel to screen

$$u = \frac{x + 0.5}{resolution_x}$$

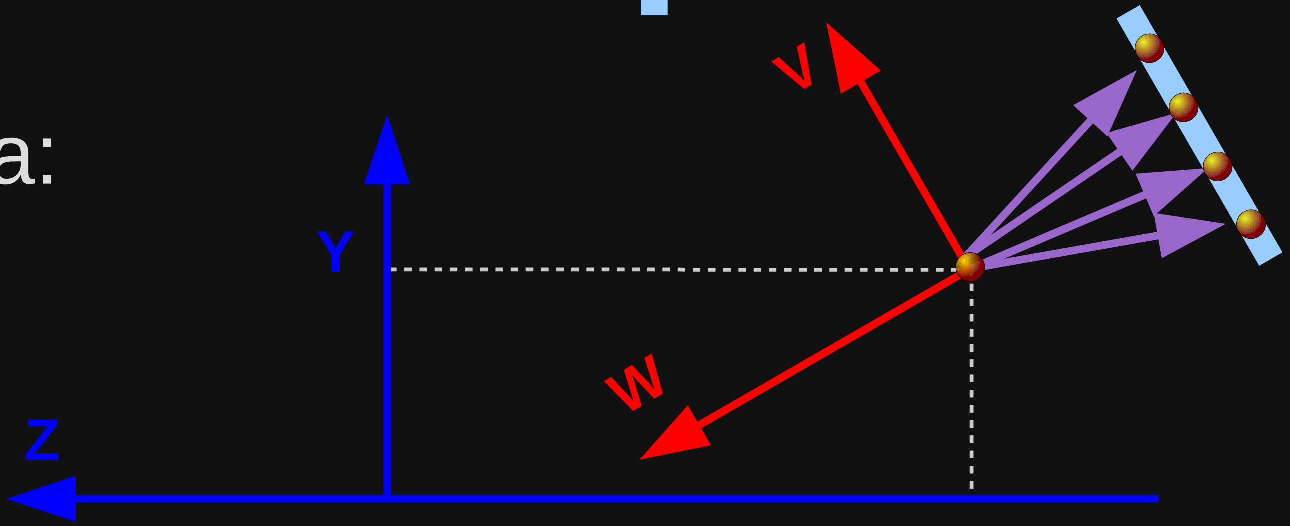
$$v = \frac{y + 0.5}{resolution_y}$$

Arbitrary Pinhole Camera

- Camera at the origin, pointing at -Z:

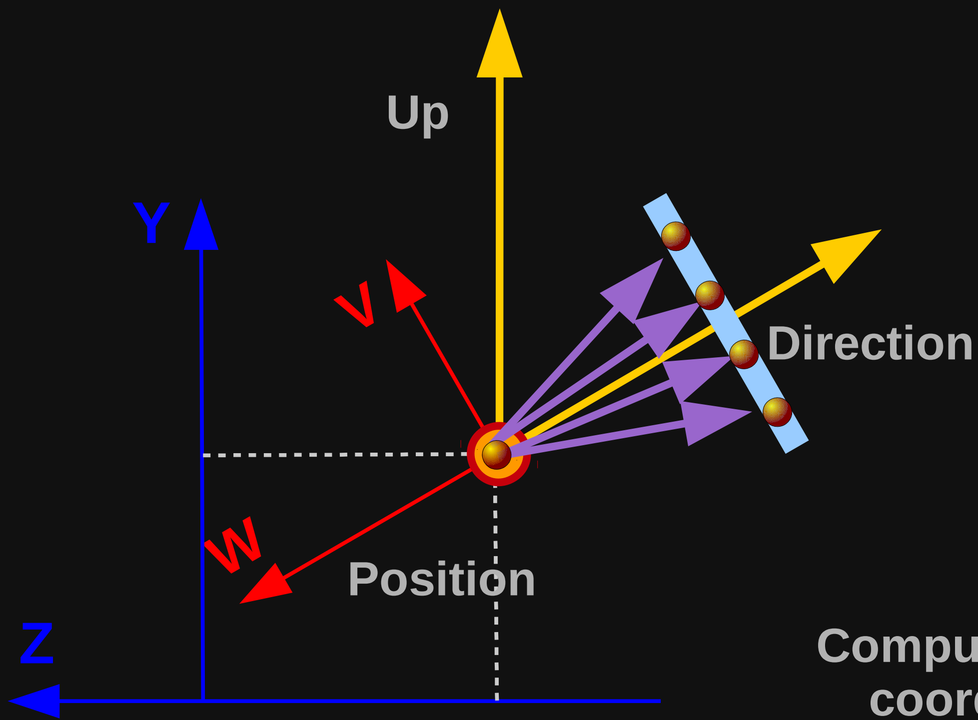


- Arbitrary camera:



Setting Up an Arbitrary Pinhole. Cam.

Constructing the camera coordinate system



Camera position: $\mathbf{p} = (x_\mathbf{p}, y_\mathbf{p}, z_\mathbf{p})$

Camera direction: $\mathbf{d} = (x_\mathbf{d}, y_\mathbf{d}, z_\mathbf{d})$

Up vector: $\mathbf{k} = (x_\mathbf{k}, y_\mathbf{k}, z_\mathbf{k})$

$$\mathbf{w} = \frac{\mathbf{d}}{|\mathbf{d}|}$$

Computing the camera's coord. System base

$$\mathbf{u} = \frac{\mathbf{k} \times \mathbf{w}}{|\mathbf{k} \times \mathbf{w}|}$$

$$\mathbf{v} = \frac{\mathbf{w} \times \mathbf{u}}{|\mathbf{w} \times \mathbf{u}|}$$

Setting Up an Arbitrary Pinhole. Cam.

- First, compute the direction of the rays in the local camera space:

$$\text{ray origin}_x = \frac{x_{pixel} + 0.5}{resolution_x} width + x_{min}$$

$$\text{ray origin}_y = \frac{y_{pixel} + 0.5}{resolution_y} height + y_{min}$$

$$\text{ray origin}_z = 0$$

$$\text{ray direction} = (0, 0, -1)$$

- Transform the ray to the world space:

$$\text{ray origin}_{\text{world}} = \mathbf{B}\, \text{ray origin}_{\text{cam}} + \mathbf{p}$$

$$\text{ray direction}_{\text{world}} = \mathbf{B}\, \text{ray direction}_{\text{cam}}$$