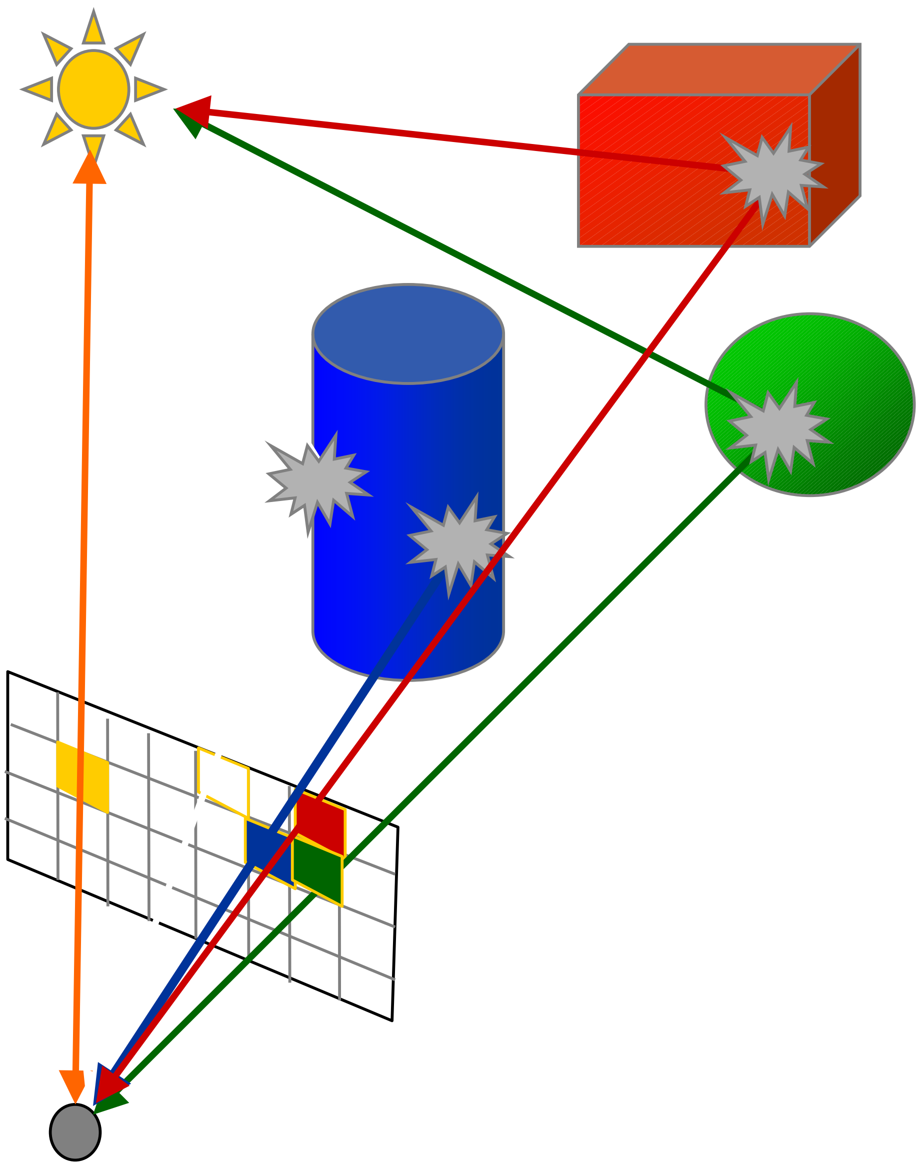
CS 563 Advanced Topics in Computer Graphics

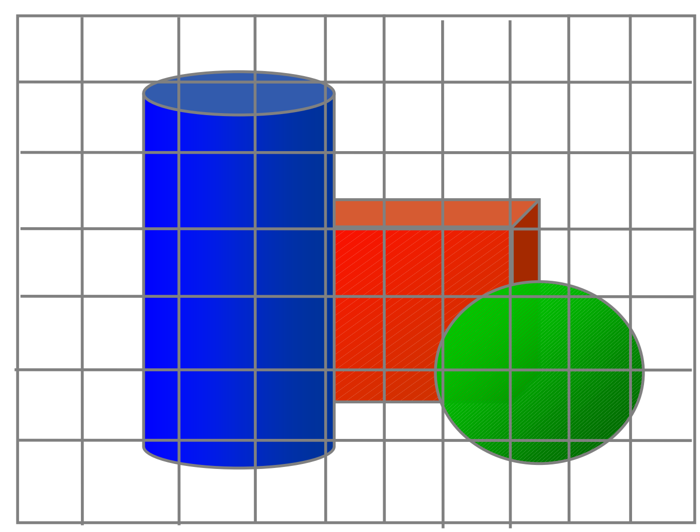
Lecture 2: Bare-Bones Raytracer

by Emmanuel Agu

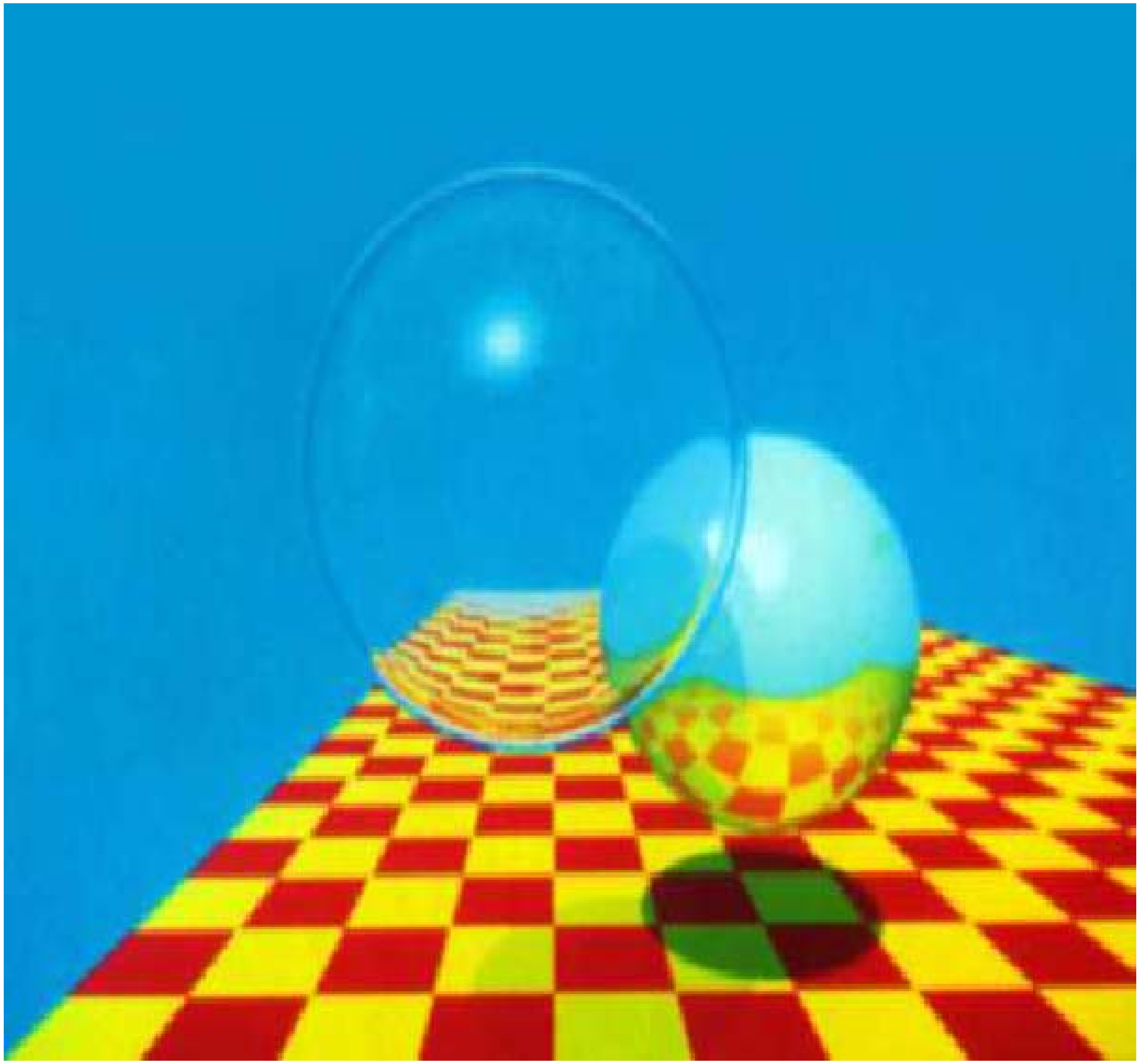
Ray Casting (Appel, 1968)



direct illumination



Recursive ray tracing (Whitted, 1980)



Pseudocode for Ray Tracer

- Basic idea

```
color Raytracer{
    for(each pixel direction){
      determine first object in this pixel direction
      calculate color shade
      return shade color
   }
}
```

More Detailed Ray Tracer Pseudocode

```
Define the objects and light sources in the scene
Set up the camera
For(int r = 0; r < nRows; r++){
  for(int c = 0; c < nCols; c++){
     1. Build the rc-th ray
     2. Find all object intersections with rc-th ray
     3. Identify closest object intersection
     4. Compute the “hit point” where the ray hits the
         object, and normal vector at that point
     5. Find  color of light to eye along ray
     6. Set rc-th pixel to this color
  }
}
```

Build the RC-th Ray

- Parametric expression ray starting at eye and passing through pixel at row r, and column c

$$ray = origin + (direction)t$$

$$r(t) = eye + dir_{rc}t$$

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

- But what exactly is this $dir_{rc}(t)$?
- need to express ray direction in terms of variables r and c
- Now need to set up camera, and then express dir_{rc} in terms of camera r and c

Building the Ray

The primary ray is defined in world coordinates by:

- The camera location $\boldsymbol{o}$ (the ray origin)
- a unit direction vector $\boldsymbol{d}$

The expression for the primary ray is

$$\boldsymbol{p} = \boldsymbol{o} + t\,\boldsymbol{d}$$

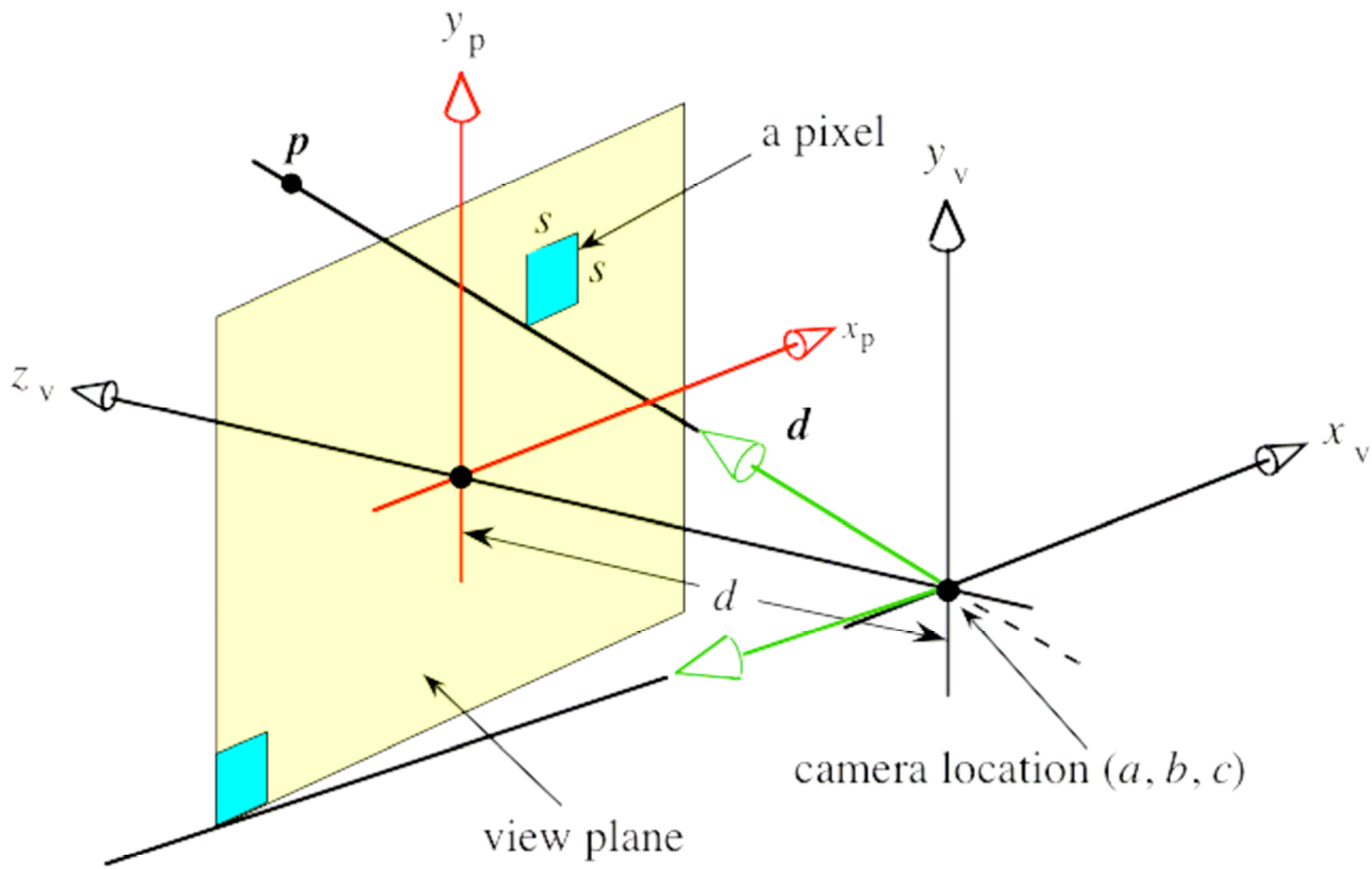
where t is the ray parameter.

Building the Ray

Note

The vector ***d*** must be converted to a unit vector before it is used in the camera ray.

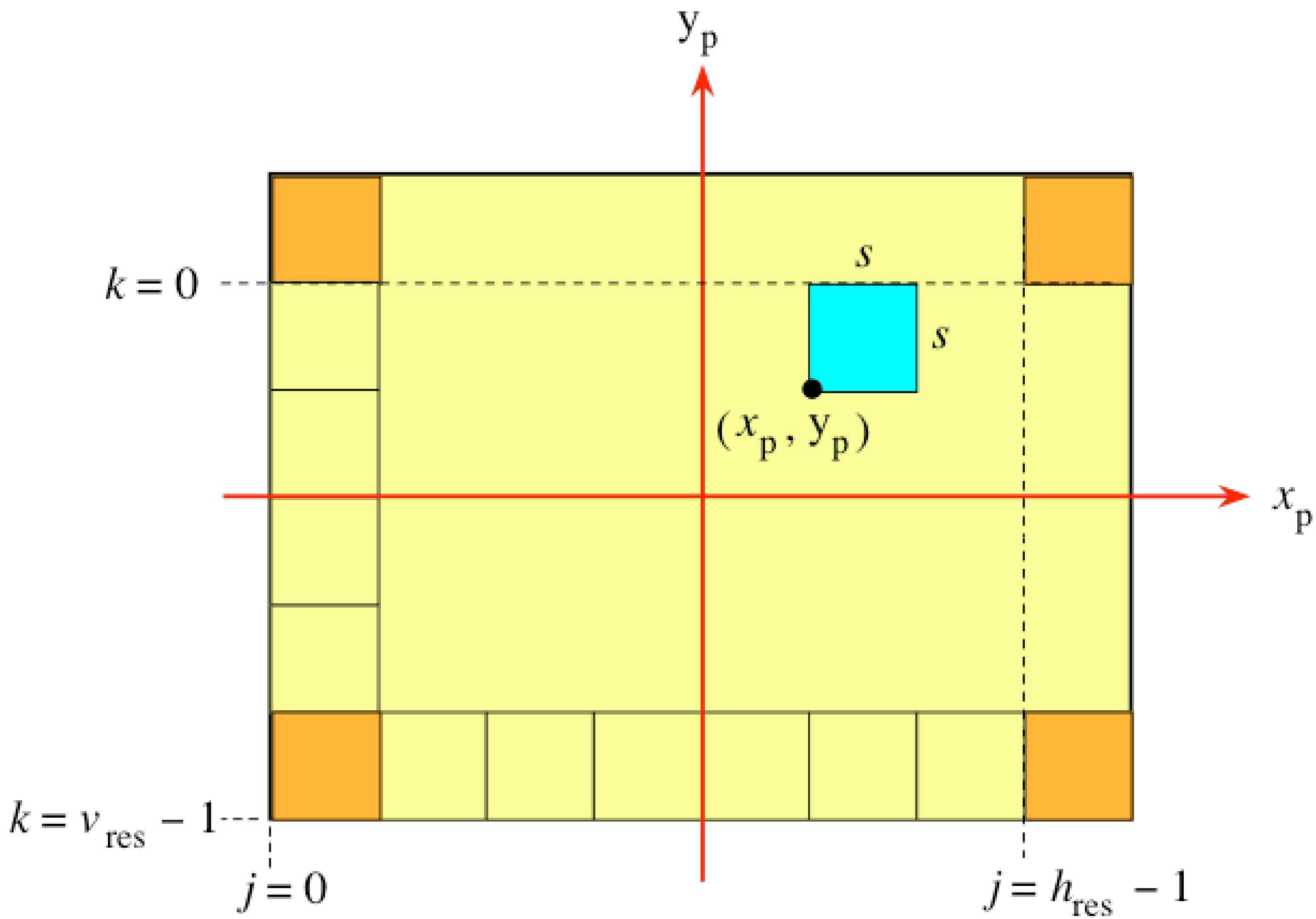
Camera: Perspective Viewing



Computer Graphics Rendering Techniques: Primary Ray Calculation © K G Suffern 1995-2002
UTS: CRICOS Provider code 0009F

5

Pixel Screen Coordinates



Calculate Pixel Position

We index the pixels horizontally (from left to right) with

$$j:\ 0 \le j \le h_{\text{res}} - 1,$$

and vertically (from top to bottom) with

$$k: 0 \le k \le v_{\text{res}} - 1.$$

The $(x_{\text{p}}, y_{\text{p}})$ coordinates of the lower left corner of the (j, k) pixel are

$$\boxed{\begin{aligned} x_{\text{p}} &= s(-h_{\text{res}} / 2 + j) \\ y_{\text{p}} &= s(v_{\text{res}} / 2 - k - 1) \end{aligned} \qquad (1)}$$

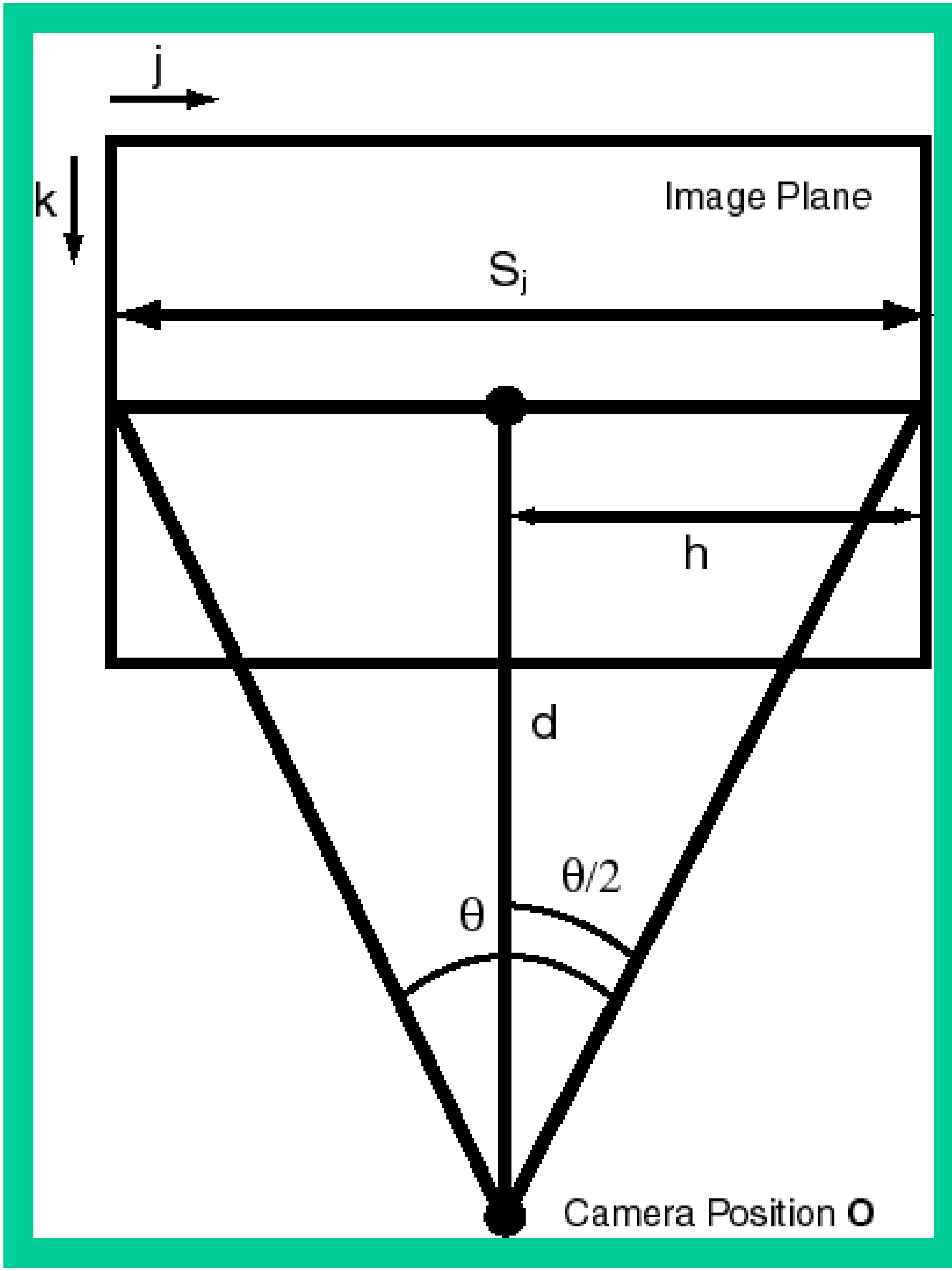
Calculating Primary Rays

- Given (in world coordinates)
  - Camera (eye point) location **O**
  - Camera view out direction ($\underline{\mathbf{Z}}_v$)
  - Camera view up vector ($\underline{\mathbf{Y}}_v$)
  - Distance to image plane (d)
  - Horizontal camera view angle (θ)
  - Pixel resolution of image plane (h_{res}, v_{res})
- Calculate set of rays (**d**) that equally samples the image plane

Calculate Preliminary Values

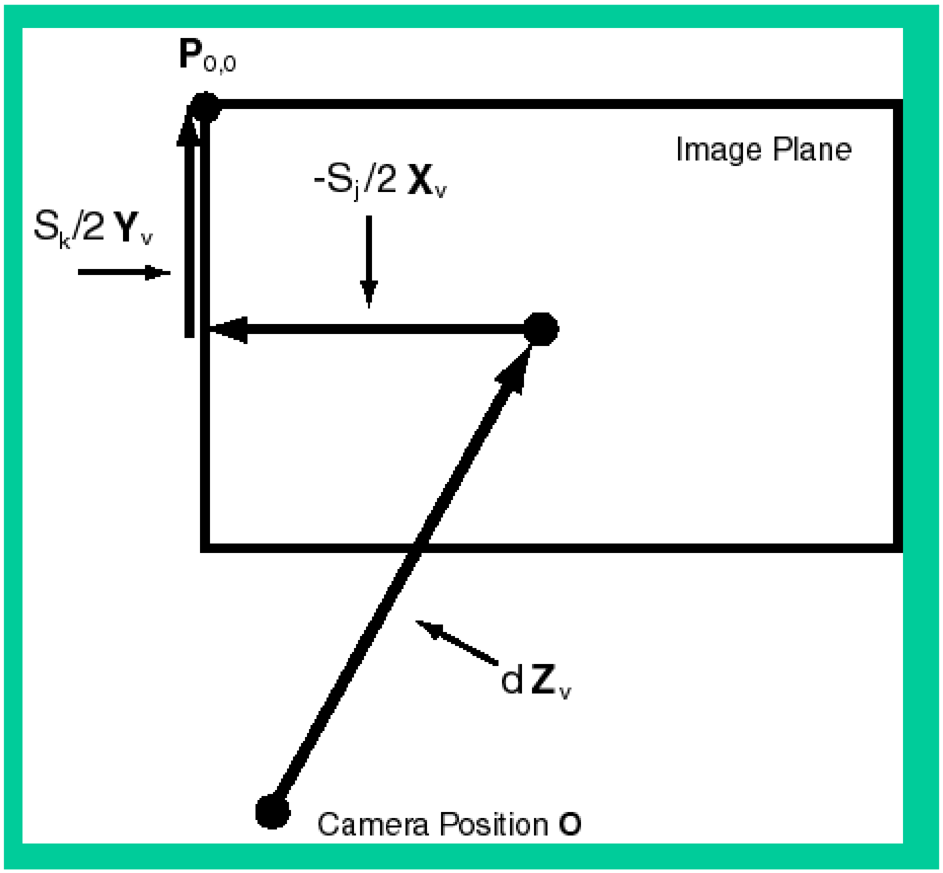
- Camera view side direction ($\underline{\mathbf{X}}_v$)
  - $\underline{\mathbf{Y}}_v \times \underline{\mathbf{Z}}_v$
- Horizontal length of image plane (s_j)
  - Next slide
- Vertical length of image plane (s_k)
  - $s_k = s_j \bullet (v_{res} / h_{res})$
  - Assume square pixels

- Calculating s_j
  - $h = d \bullet \tan(\theta/2)$
  - $s_j = 2h$
  - $s_j = 2d \bullet \tan(\theta/2)$



Calculate Preliminary Values

- Position of top left pixel ($\underline{\mathbf{P}}_{0,0}$)
  - $\underline{\mathbf{O}} + d * \underline{\mathbf{Z}}_v - (S_j/2) * \underline{\mathbf{X}}_v + (S_k/2) * \underline{\mathbf{Y}}_v$



All in world coordinates!

Calculate Those Rays!

- $\underline{\mathbf{P}}_{0,0} + \alpha\, \underline{\mathbf{X}}_v - \beta\, \underline{\mathbf{Y}}_v$ sweeps out image plane
- $0 \leq \alpha \leq S_{j;}\ 0 \leq \beta \leq S_k$

for (j=0; j++; j < h_{res})
    for (k=0; k++; k < v_{res}) {
        $\underline{\mathbf{d}}_{j,k} = (\underline{\mathbf{P}}_{0,0} + S_j * (j/(h_{res}-1)) * \underline{\mathbf{X}}_v$
                $- S_k * (k/(v_{res}-1)) * \underline{\mathbf{Y}}_v) - \underline{\mathbf{O}};$
        $\underline{\mathbf{d'}}_{j,k} = \underline{\mathbf{d}}_{j,k} \,/\, |\, \underline{\mathbf{d}}_{j,k} \,|\ ;$
        Image[j,k] = ray_trace($\underline{\mathbf{O}}$, $\underline{\mathbf{d'}}_{j,k}$, Scene);
    }

Perspective projection

In ray tracing we do not need to explicitly use the perspective projection, because it is built into the primary rays.

As these emanate from camera position, this acts as the centre of projection.

Size of the pixels

What effect does changing the physical size of the pixels have on the image?

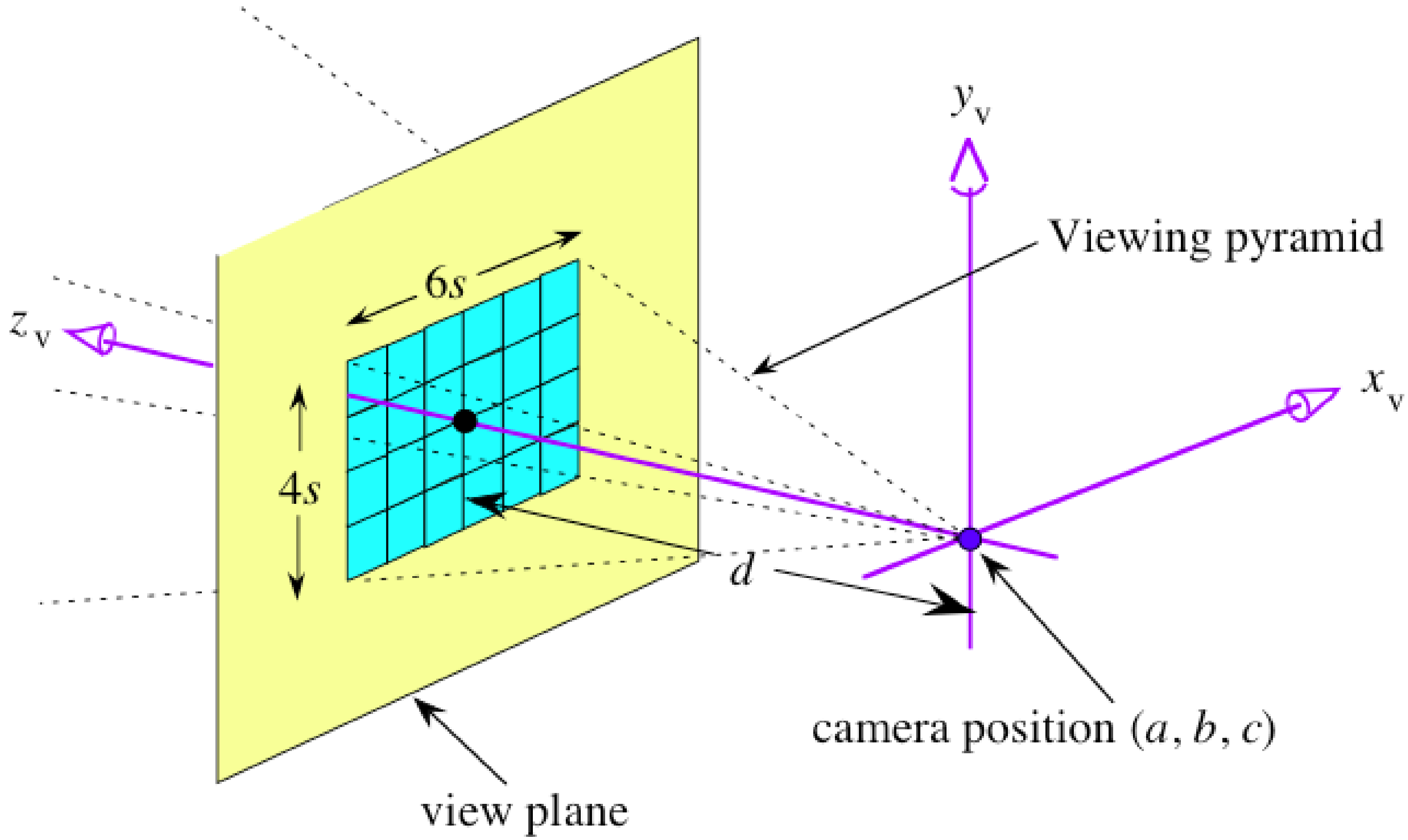
For a specified image resolution (h_{res}, v_{res}), the size of the window is proportional to the size s of the pixels.

For a fixed view plane distance d, the field of view is proportional to s.

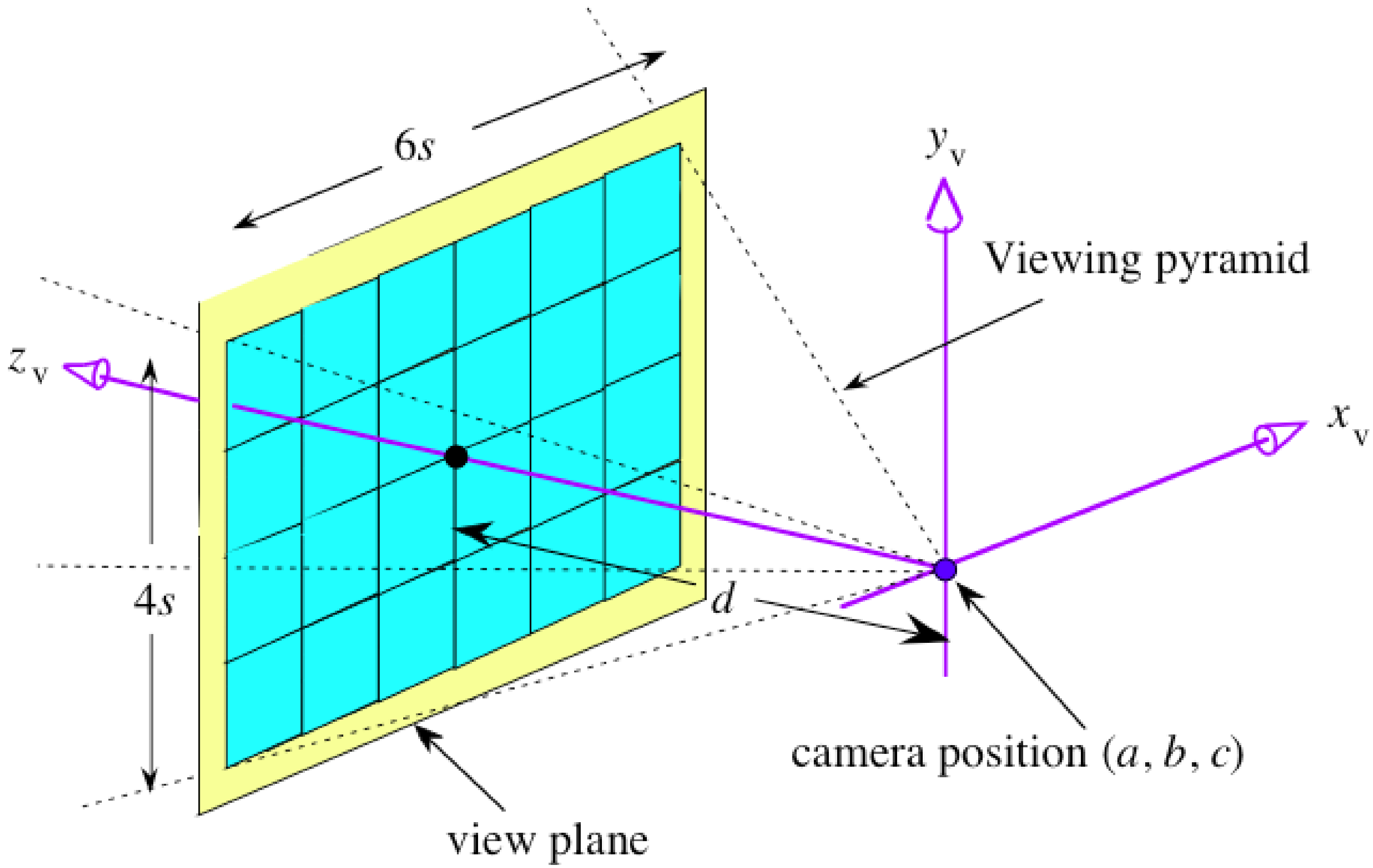
For fixed s, the size of the window is proportional to h_{res} and v_{res}.

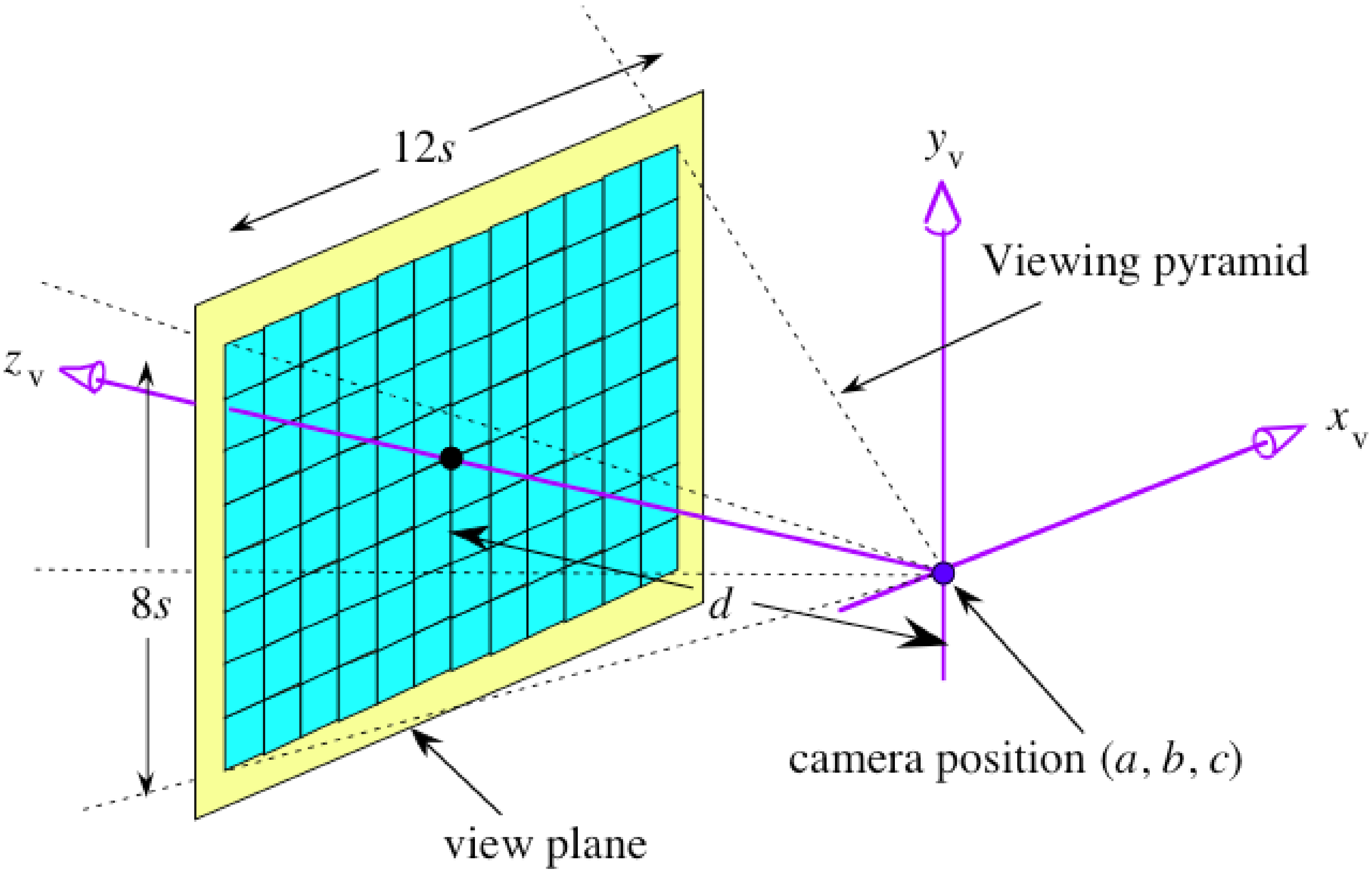
Increasing h_{res} and v_{res} increases the *field of view* of the camera, provided s and d are kept the same.

Some of these effects are illustrated in the following figures.



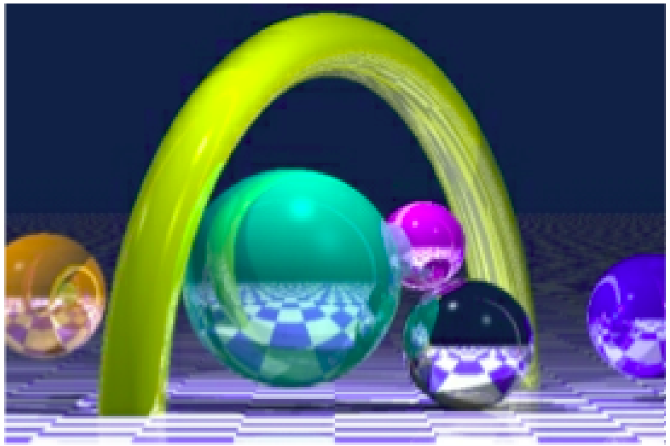
y_v
Viewing pyramid
z_v
$6s$
$4s$
x_v
d
camera position (a, b, c)
view plane


6s
y_v
Viewing pyramid
z_v
x_v
4s
d
camera position (a, b, c)
view plane

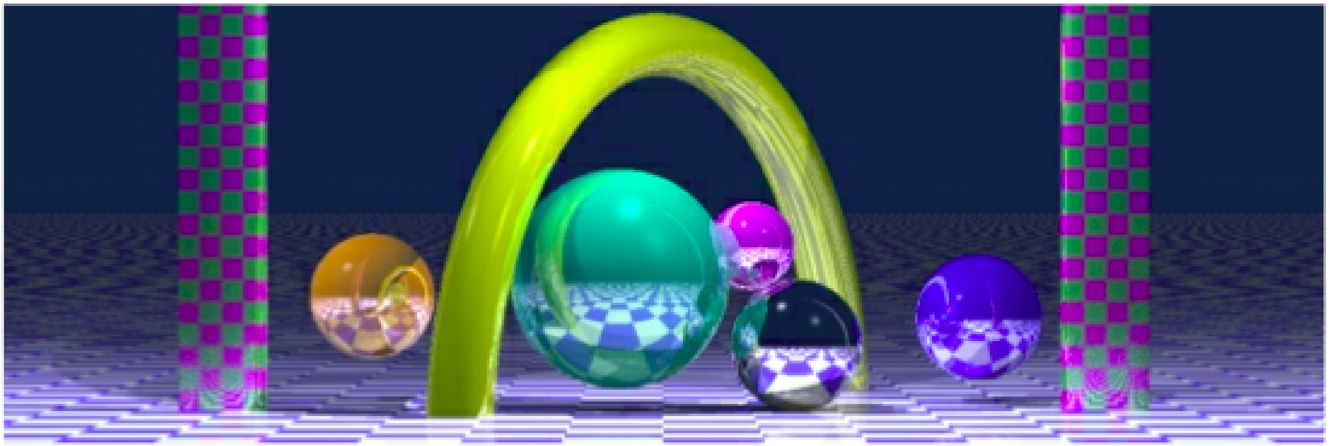

12s
y_v
Viewing pyramid
z_v
x_v
8s
d
camera position (a, b, c)
view plane

The following three foils illustrate the effect of changing the image resolution on the field of view, when the pixel size is kept the same.

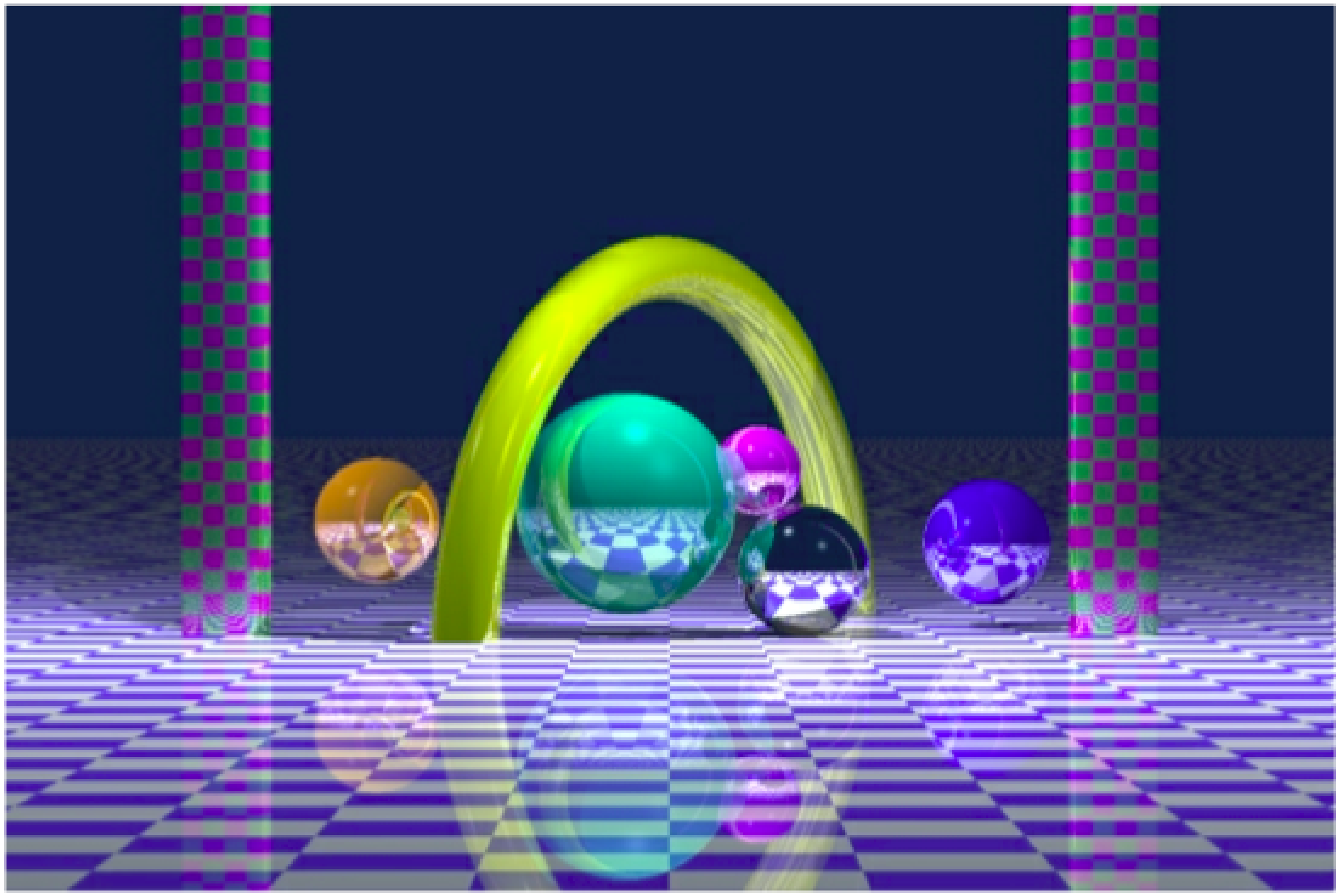
It is common practice in ray tracers to specify the field of view with angles. You can still do this with appropriate conversions.



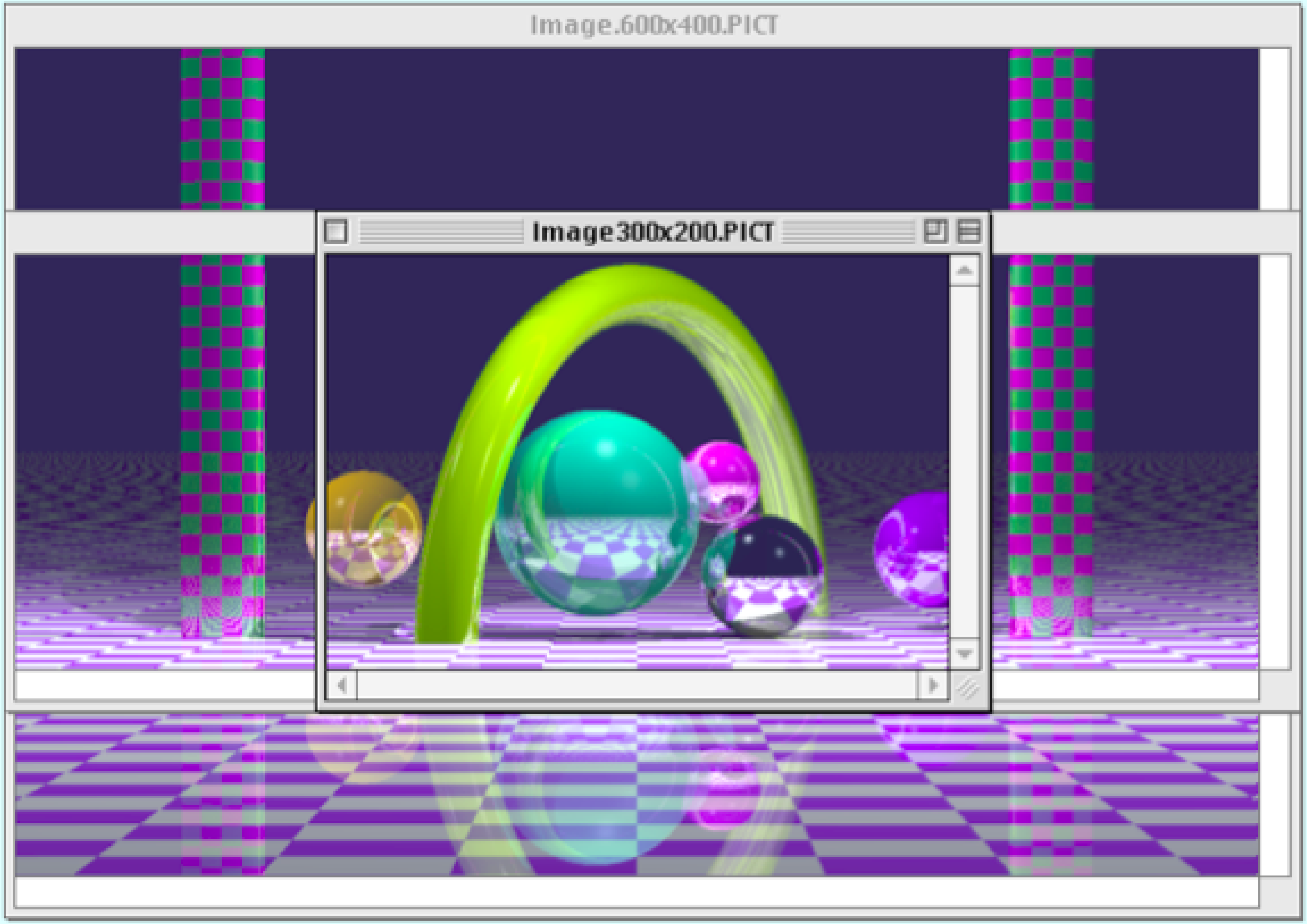
300 x 200 pixels



600 x 200 pixels



600 x 400 pixels



The three windows superimposed

Parameters

- X and Y resolution of image
- Camera location & direction
- Distance between camera & image plane
- Camera view angle
- Distance between pixels
- These are not independent!
- Goal $\rightarrow$ Choose your independent variables and calculate your **d**'s

I recommend setting …

- X and Y resolution of image
  - (h_{res}, v_{res})
- Camera location & orientation
  - $\underline{\mathbf{O}}$ & $\underline{\mathbf{Z}}_v$ & $\underline{\mathbf{Y}}_v$
- Distance between camera & image plane
  - d (a positive scalar, e.g. 10)
- Camera view angle
  - θ

Find Object Intersections with rc-th ray

- Much of work in ray tracing lies in finding intersections with generic objects
- Break into two parts
  - Deal with untransformed, generic (dimension 1) shape
  - Then embellish to deal with transformed shape
- Ray generic object intersection best found by using implicit form of each shape. E.g. generic sphere is

$$F(x, y, z) = x^2 + y^2 + z^2 - 1$$

- Approach: ray r(t) hits a surface when its implicit eqn = 0
- So for ray with starting point ***o*** and direction ***d***

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

$$F(\mathbf{o} + \mathbf{d}t_{hit}) = 0$$

Ray Intersection with Generic Plane

- Generic Plane?
- Yes! Floors, walls, in a room, etc
- Generic plane is *xy*-plane, or $z = 0$
- For ray

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

- There exists a t_{hit} such that

$$\mathbf{o}_z + \mathbf{d}_z t_h = 0$$

- Solving,

$$t_{hit} = -\frac{\mathbf{o}_z}{\mathbf{d}_z}$$

Ray Intersection with Generic Plane

- Hit point P_{hit} is given by

$$P_{hit} = \mathbf{o} - \mathbf{d}(\mathbf{o}_z / \mathbf{d}_z)$$

- Numerical example?
- Where does the ray $r(t) = (4, 1, 3) + (-3, -5, -3)t$ hit the generic plane?
- Soln:

$$t_{hit} = -\frac{\mathbf{o}_z}{\mathbf{d}_z} = -\frac{3}{-3} = 1$$

- And hit point is given by

$$\mathbf{o} + \mathbf{d} = (1, -4, 0)$$

Ray-Sphere Intersection

G. Drew Kessler

Larry Hodges

Georgia Institute of Technology

Ray/Sphere Intersection (Algebraic Solution)

Ray is defined by $R(t) = R_o + R_d * t$ where $t > 0$.

R_o = Origin of ray at (x_o, y_o, z_o)

R_d = Direction of ray $[x_d, y_d, z_d]$ (unit vector)

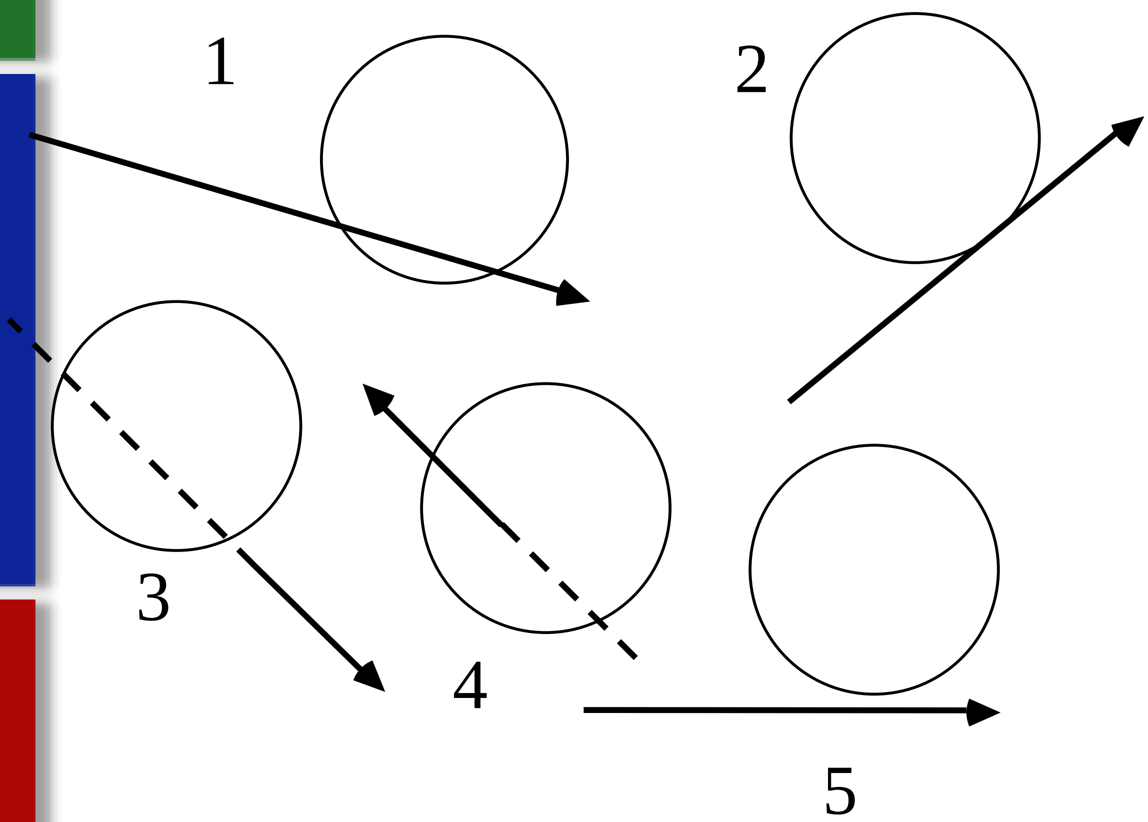
Sphere's surface is defined by the set of points $\{(x_s, y_s, z_s)\}$ satisfying the equation:

$$(x_s - x_c)^2 + (y_s - y_c)^2 + (z_s - z_c)^2 - r_s^2 = 0$$

Center of sphere: (x_c, y_c, z_c)

Radius of sphere: r_s

Possible Cases of Ray/Sphere Intersection



1. Ray intersects sphere twice with t>0
2. Ray tangent to sphere
3. Ray intersects sphere with t<0
4. Ray originates inside sphere
5. Ray does not intersect sphere

Solving For t

Substitute the basic ray equation:

$x = x_o + x_d*t$

$y = y_o + y_d*t$

$z = z_o + z_d*t$

into the equation of the sphere:

$(x_0 + x_dt - x_c)^2 + (y_0 + y_dt - y_c)^2 + (z_0 + z_dt - z_c)^2 - r_s^2 = 0$

This is a quadratic equation in t: $At^2 + Bt + C = 0$, where

$A = x_d^2 + y_d^2 + z_d^2$

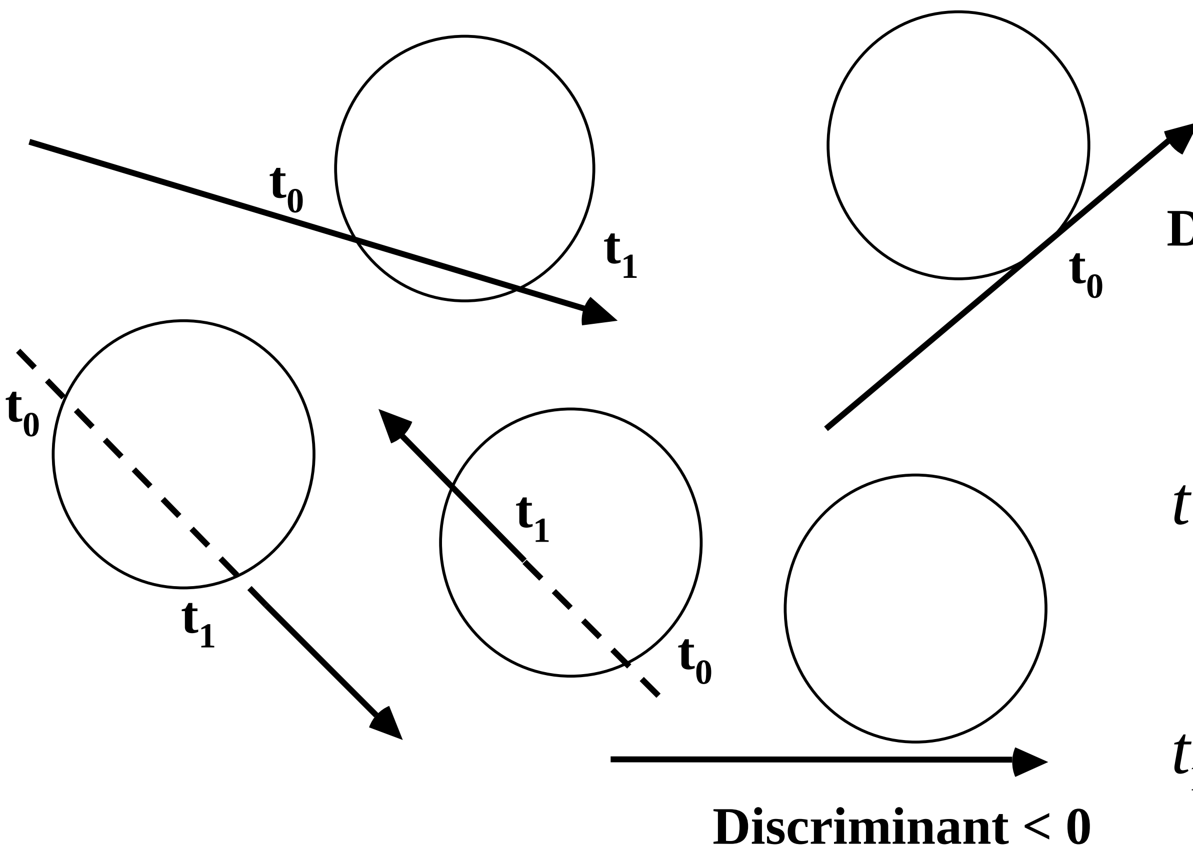
$B = 2[x_d(x_0 - x_c) + y_d(y_0 - y_c) + z_d(z_0 - z_c)]$

$C = (x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2 - r_s^2$

Note: A=1

Relation of t to Intersection

We want the smallest positive t - call it t_i



$$t_0 = \left(\frac{-B - \sqrt{B^2 - 4AC}}{2} \right)$$

$$t_1 = \left(\frac{-B + \sqrt{B^2 - 4AC}}{2} \right)$$

Actual Intersection

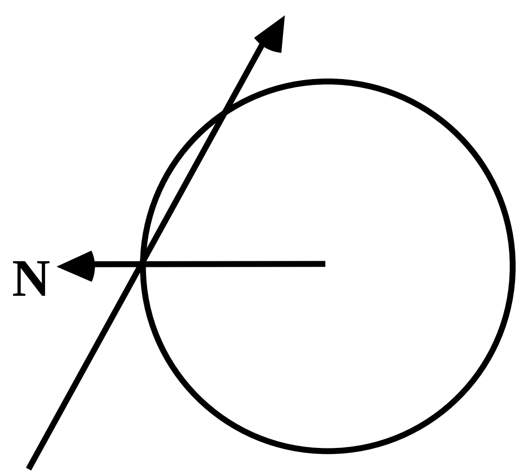
Intersection point,

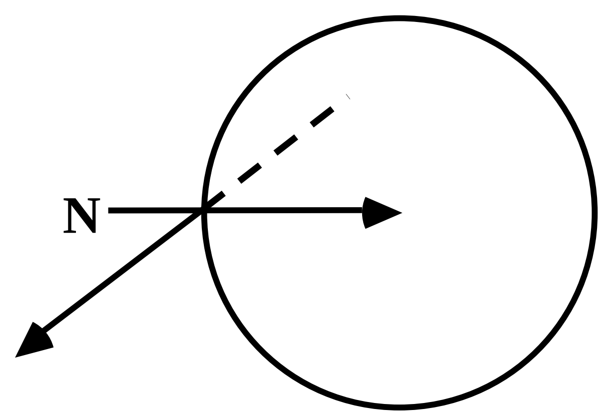
$$(x_i, y_i, z_i) = (x_o+x_d*t_i, y_o+y_d*t_i, z_o+z_d*t_i)$$

Unit vector normal to the surface at this point is

$$N = [(x_i - x_c) / r_s, (y_i - y_c) / r_s, (z_i - z_c) / r_s]$$

If the ray originates inside the sphere, N should be negated so that it points back toward the center.





Summary (Algebraic Solution)

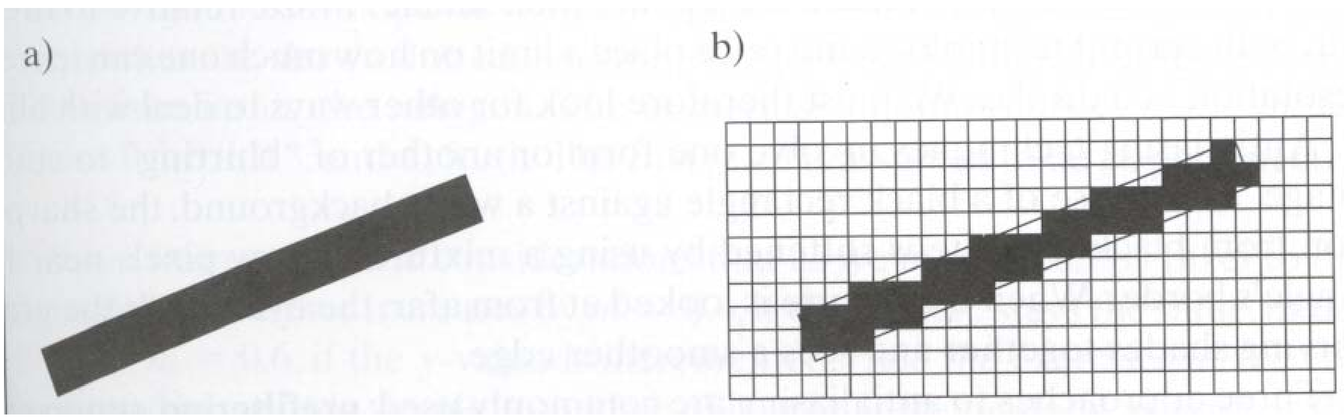
1. Calculate A, B and C of the quadratic intersection equation
2. Calculate discriminant (If < 0, then no intersection)
3. Calculate t_0
4. If $t_0 < 0$, then calculate t_1 (If $t_1 < 0$, no intersection point on ray)
5. Calculate intersection point
6. Calculate normal vector at point

Helpful pointers:

- Precompute r_s^2
- Precompute $1/r_s$
- If computed t is very small then, due to rounding error, you may not have a valid intersection

Antialiasing

- Raster displays have pixels as rectangles
- Aliasing: Discrete nature of pixels introduces "jaggies"



Antialiasing

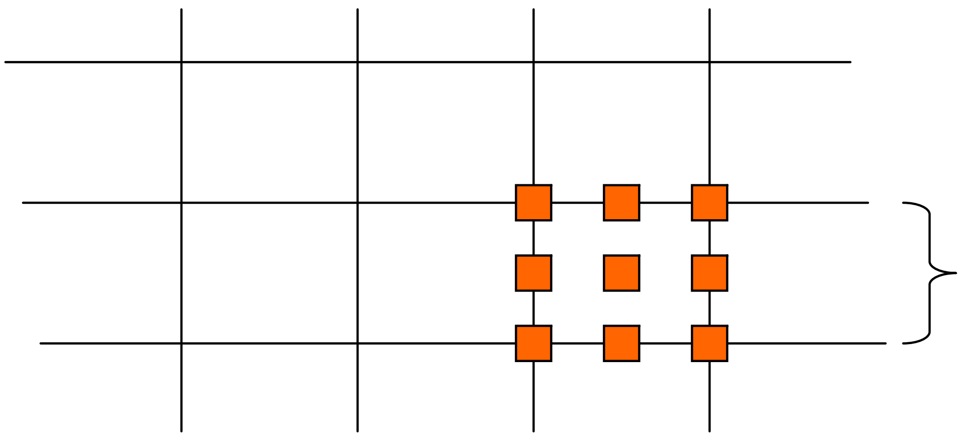
- Aliasing effects:
  - Distant objects may disappear entirely
  - Objects can blink on and off in animations
- Antialiasing techniques involve some form of blurring to reduce contrast, smoothen image
- Three antialiasing techniques:
  - Prefiltering
  - Postfiltering
  - Supersampling

Prefiltering

- Basic idea:
  - compute area of polygon coverage
  - use proportional intensity value
- Example: if polygon covers ¼ of the pixel
  - use ¼ polygon color
  - add it to ¾ of adjacent region color
- Cons: computing pixel coverage can be time consuming

Supersampling

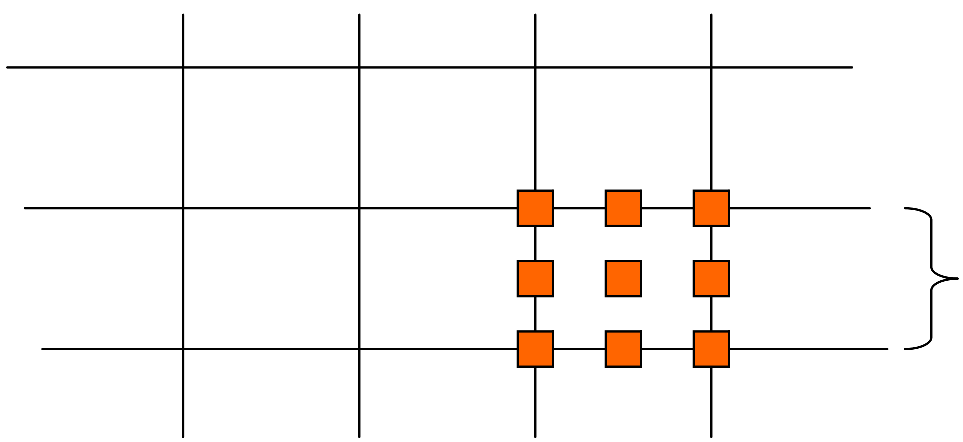
- Useful if we can compute color of any (x,y) value on the screen
- Increase frequency of sampling
- Instead of (x,y) samples in increments of 1
- Sample (x,y) in fractional (e.g. ½) increments
- Find average of samples
- Example: Double sampling = increments of ½ = 9 color values averaged for each pixel



Average 9 (x, y) values to find pixel color

Postfiltering

- Supersampling uses average
- Gives all samples equal importance
- Post-filtering: use weighting (different levels of importance)
- Compute pixel value as weighted average
- Samples close to pixel center given more weight



Sample weighting

1/16	1/16	1/16
1/16	1/2	1/16
1/16	1/16	1/16

References / Shamelessly stolen

- Kevin Suffern, Ray Tracing from the Ground up
- David Breen, Drexel University CS 431/636 Advanced Rendering Techniques
- Hill and Kelley, Computer Graphics using OpenGL, 3rd edition