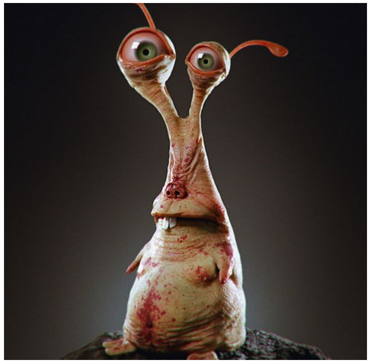
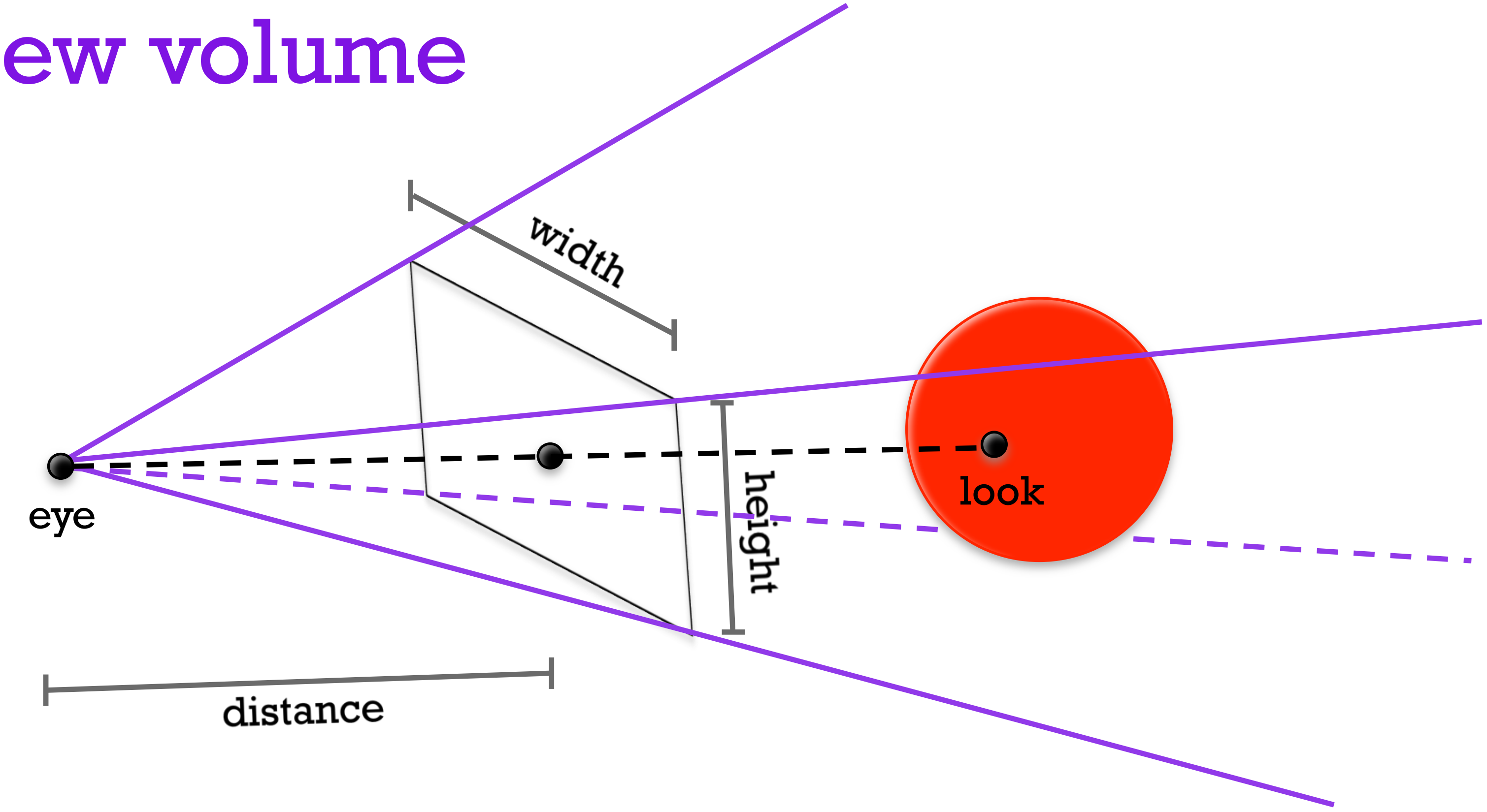
3D Rendering

3D Computer Graphics (Lab 4)



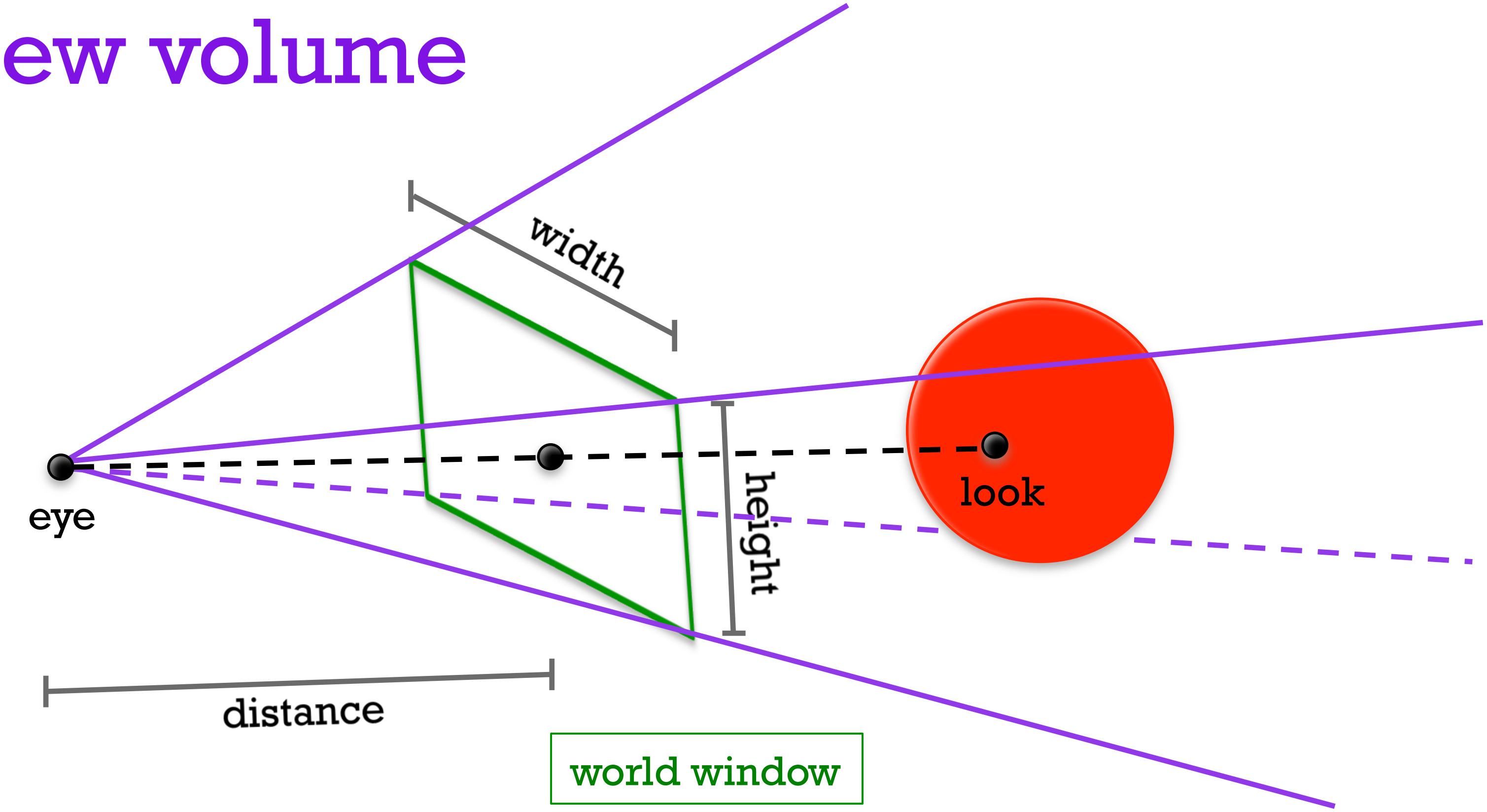
Which portion of the 3D scene is shown in the final image?

View volume



- View volume = a portion of the 3D space which is represented in the final image.
- 3D objects are only rendered if they lie inside the view volume.
- The view volume is defined by 3 real numbers: distance, width and height.

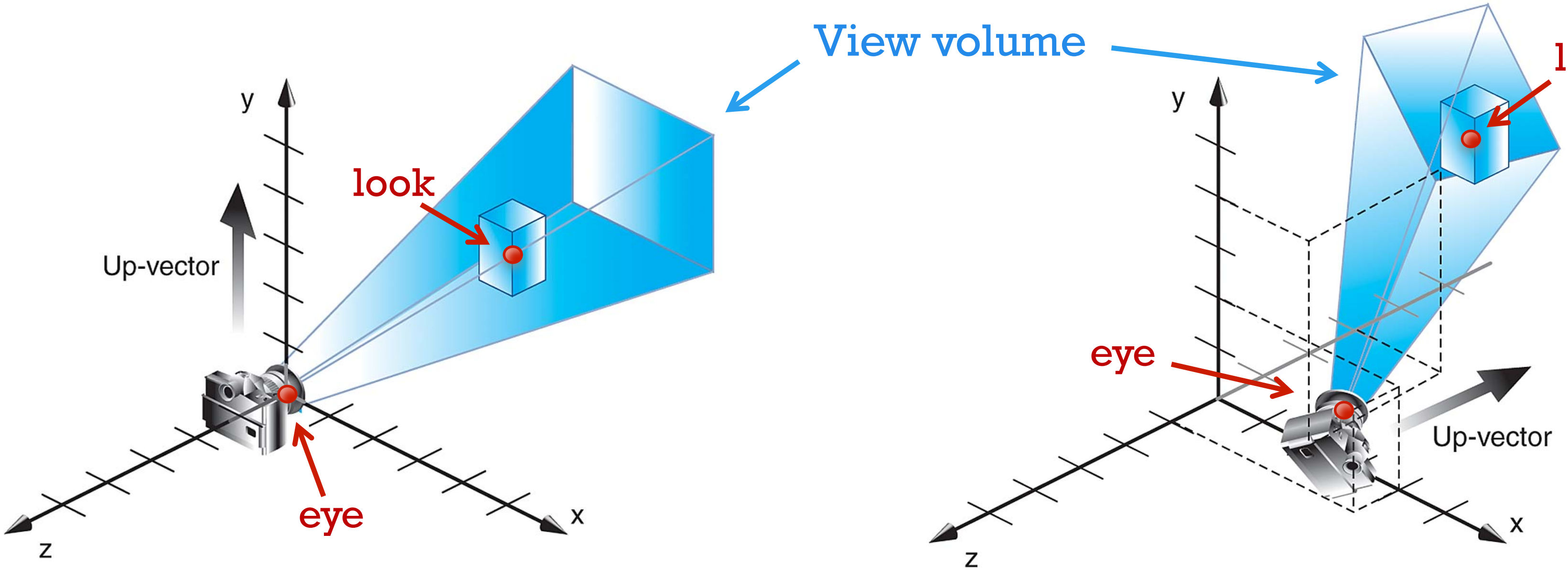
View volume



- View volume = a portion of the 3D space which is represented in the final image.
- 3D objects are only rendered if they lie inside the view volume.
- The view volume is defined by 3 real numbers: distance, width and height.

Remember …

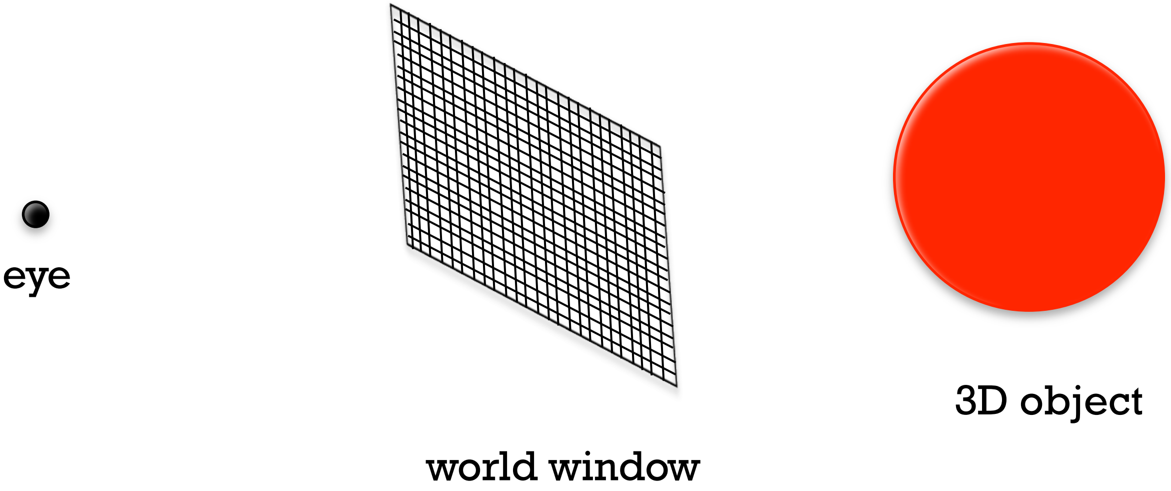
- An *eye* point indicating the location of the camera.
- A reference point (*look*) indicating the point the camera is aimed at.
- An *up vector* indicating the upwards direction of the camera.



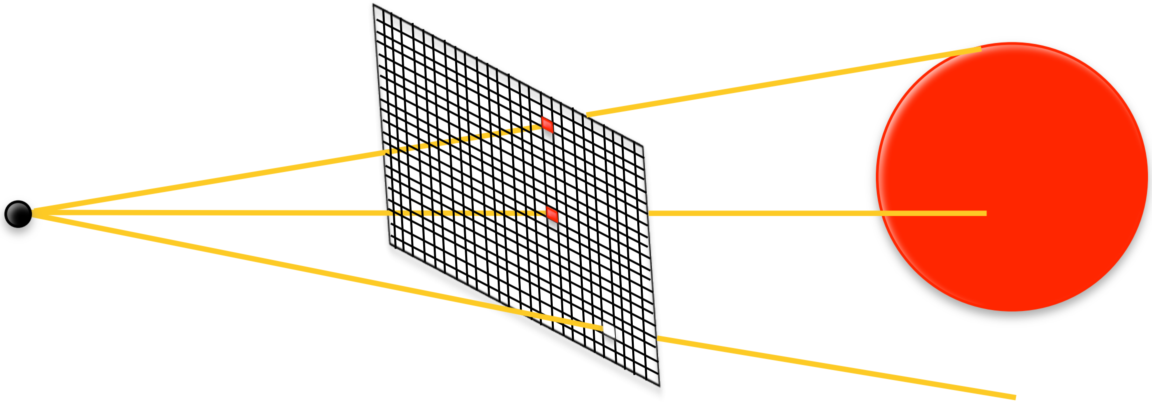
- OpenGL: gluLookAt(eye.x, eye.y, eye.z, look.x, look.y, look.z, up.x, up.y, up.z);

How can we create a 2D image from the 3D object taking into account the camera properties?

Idea …

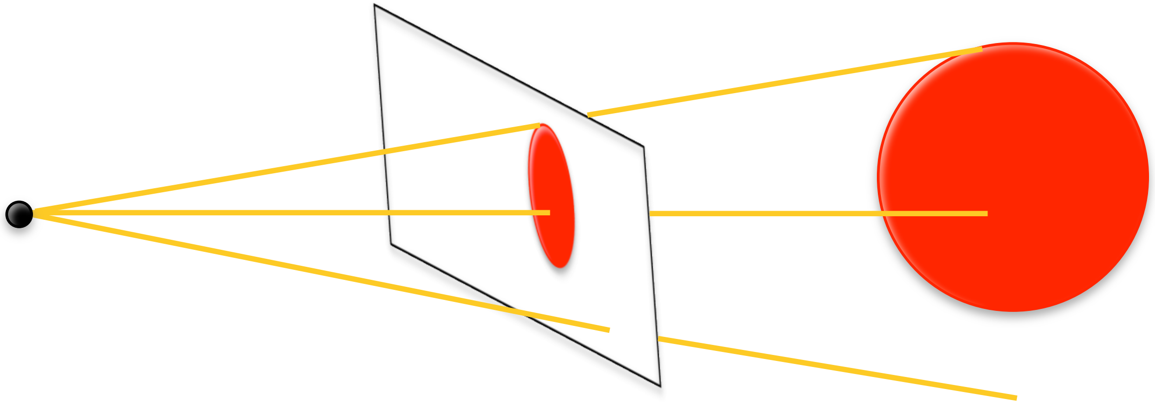


Idea …



Idea …

Ray tracing



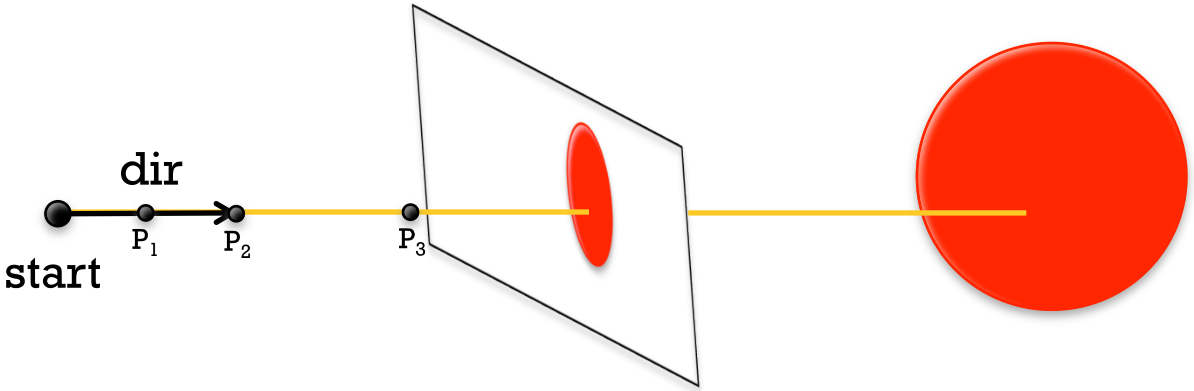
For each pixel

1. Create a ray from the eye of the camera through this pixel.
2. Compute the intersection of this ray with the 3D object. The colour of the intersection point determines the colour of the pixel.

Ray

- A ray is defined by a point and a vector:
  - start the point where the ray originates,
  - dir the direction of the ray.

It is not necessary to normalize dir.



t	start + t.dir
0	start
0.5	start + 0.5dir = P_1
1	start + dir = P_2
2	start + 2dir = P_3

For every point P on the ray, there exists a positive real number t such that

P = start + t.dir

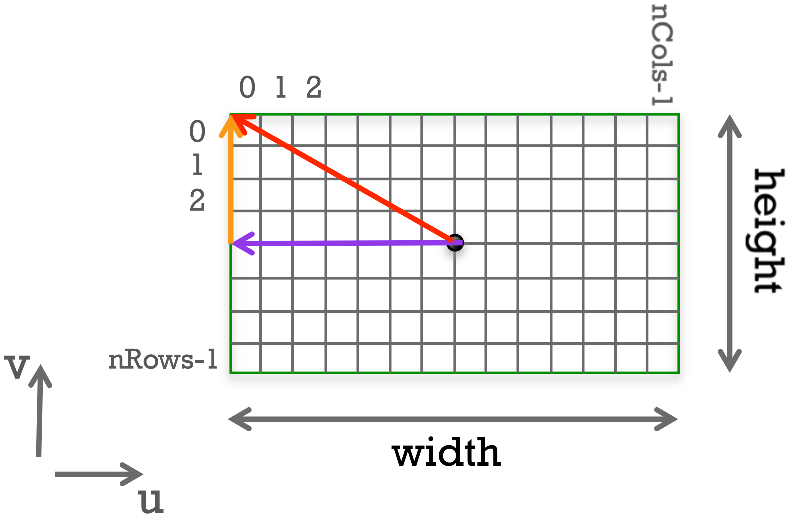
Pixel position

What is the width of a pixel?

widthPixel = width/nCols

What is the height of a pixel?

heightPixel = height/nRows

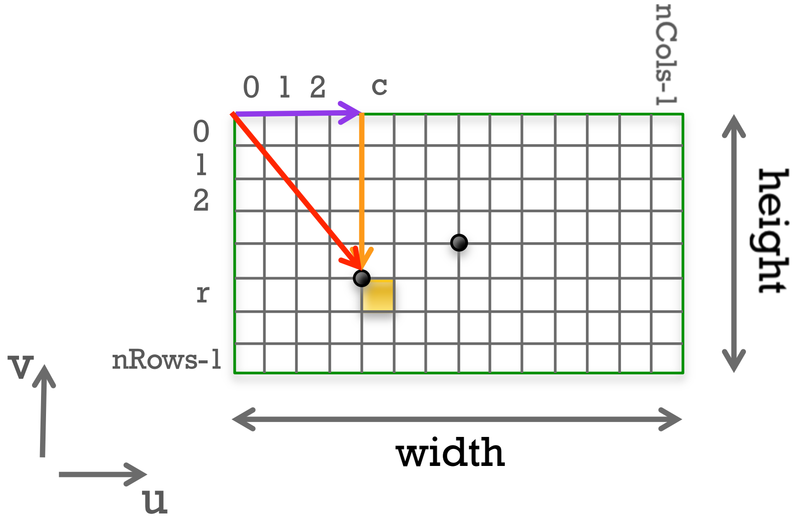


What does the following line of code do?

Vector v1 = -(width/2)*u + (height/2)*v

It gives a vector from the center of the image to the upper left corner of the image.

Pixel position



What is the width of a pixel?

widthPixel = width/nCols

What is the height of a pixel?

heightPixel = height/nRows

What does the following line of code do?

Vector v1 = -(width/2)*u + (height/2)*v

It gives a vector from the center of the image to the upper left corner of the image.

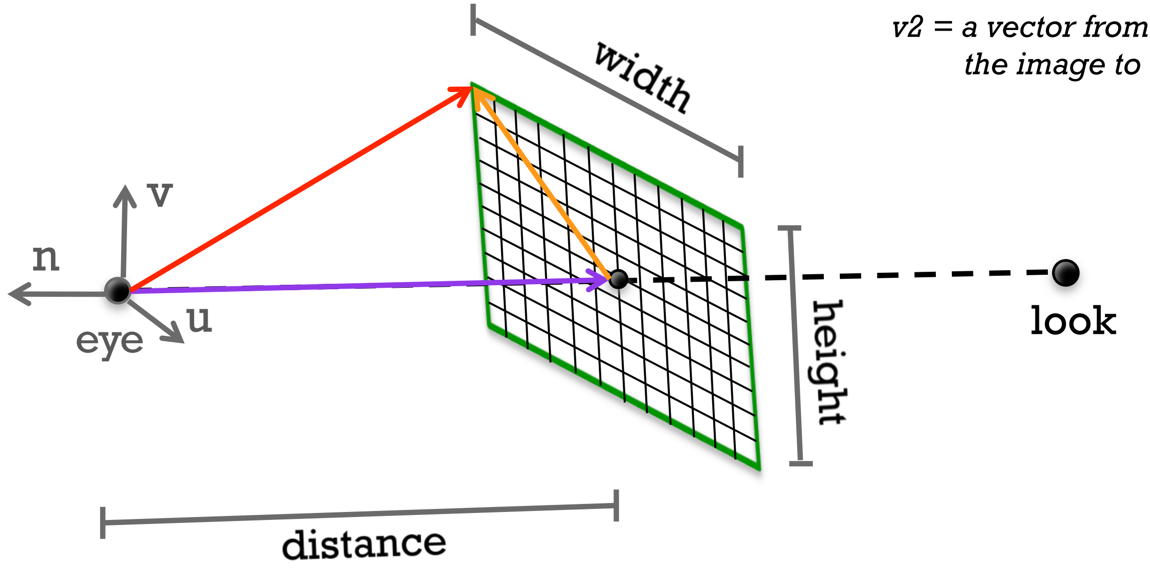
How can we find the vector from the upper left corner of the image to (the upper left corner of) a random pixel (r,c)?

Vector v2 = (c*widthPixel)*u -(r*heightPixel)*v

Ray direction

v1 = a vector from the center of the image to the upper left corner of the image.

v2 = a vector from the upper left corner of the image to a random pixel (r,c)



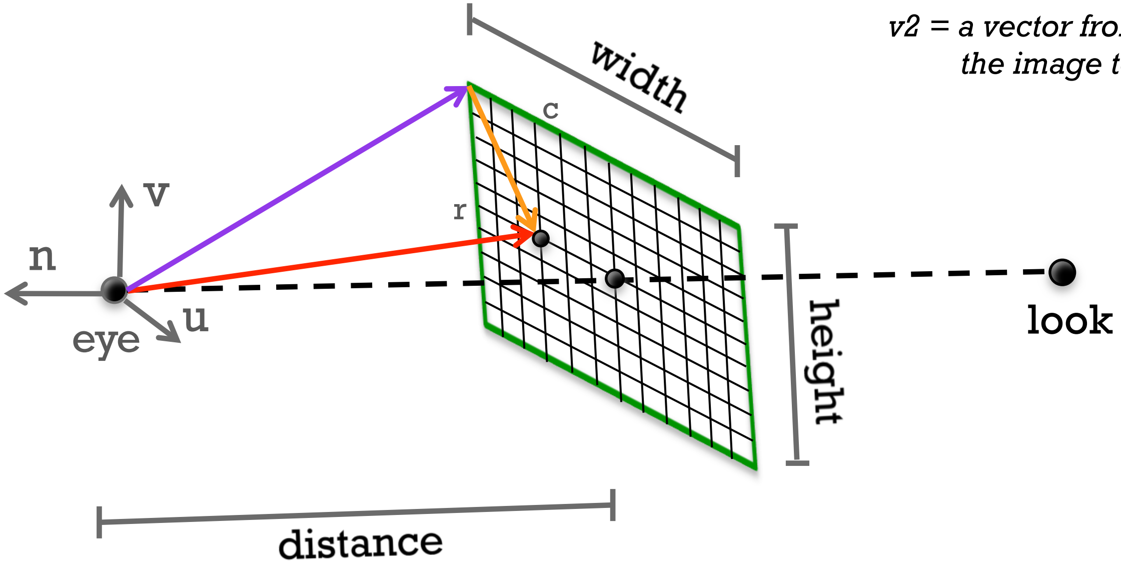
How can we find the vector from the eye to the upper left corner of the image?

Vector $\boxed{v3}$ = $\boxed{\text{-distance*n}}$ + $\boxed{v1}$

Ray direction

v1 = a vector from the center of the image to the upper left corner of the image.

v2 = a vector from the upper left corner of the image to a random pixel (r,c)



How can we find the vector from the eye to the upper left corner of the image?

Vector v3 = -distance*n + v1

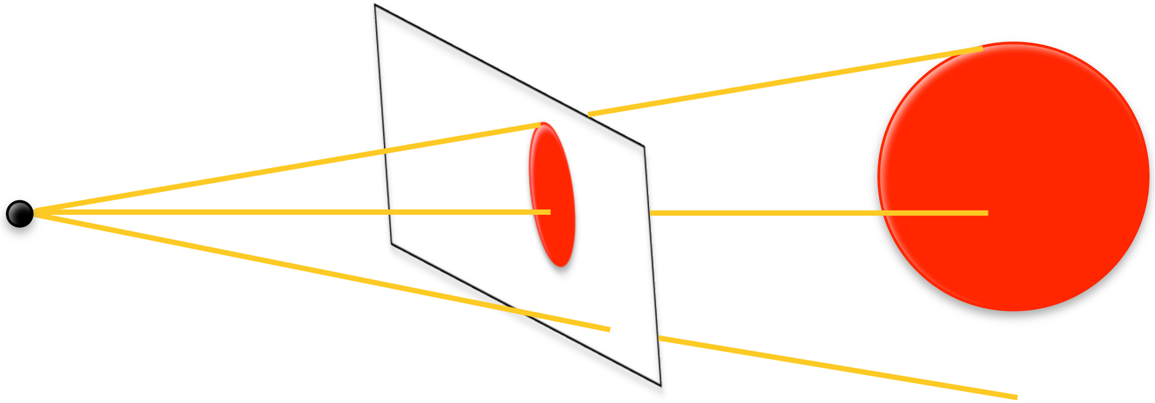
How can we find the vector from the eye to the (upper left corner of) a random pixel (r,c)?

Vector dir = v3 + v2

Ray tracing

For each pixel

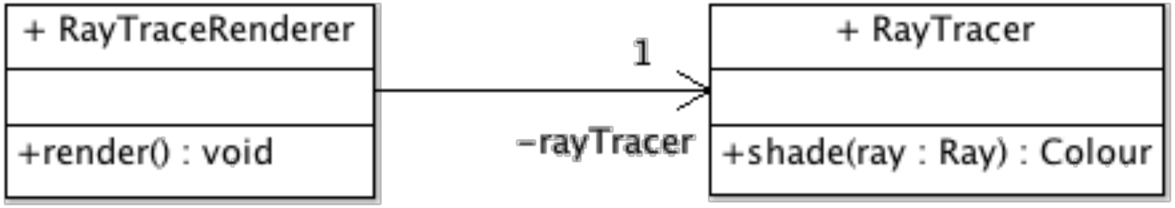
1. Create a ray from the eye of the camera through this pixel.
2. Compute the intersection of this ray with the 3D object. The colour of the intersection point determines the colour of the pixel.



Ray tracing

For each pixel

1. Create a ray from the eye of the camera through this pixel.
2. Compute the intersection of this ray with the 3D object. The colour of the intersection point determines the colour of the pixel.



render() : void

```
Ray ray = new Ray(eye)
Colour col = new Colour()
Vector v3 = vector from eye to upper left corner of image

for r from 0 to nRows-1
    for c from 0 to nCols-1
            ray.dir =  vector from the eye to pixel (r,c)
            col.set(rayTracer.shade(ray))
            // openGL commands which set
            // the colour of pixel (r,c) to col
```

shade(ray : Ray) : Colour

computes the intersection of the given ray with the 3D object.

returns the colour of this intersection point.

Ray tracing

shade(ray : Ray) : Colour

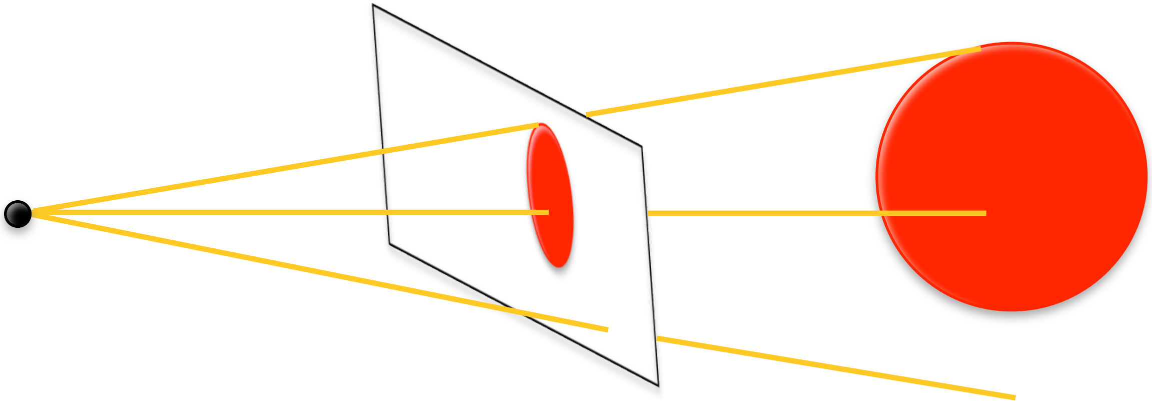
computes the intersection of the given ray with the 3D object.

returns the colour of this intersection point.

There are several issues …

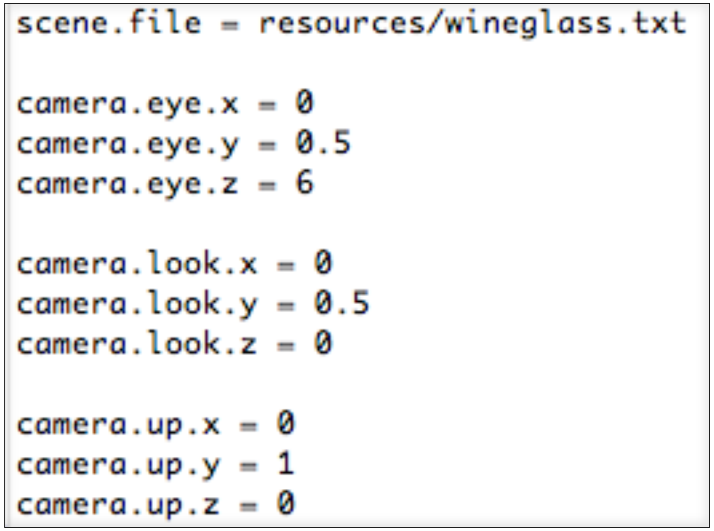
- What colour is returned if the ray does not hit the 3D object?
- Which class will carry out the intersection calculations?
- Are we only going to render scenes with one object?

Dealing with these issues calls for several structural changes to our rendering framework



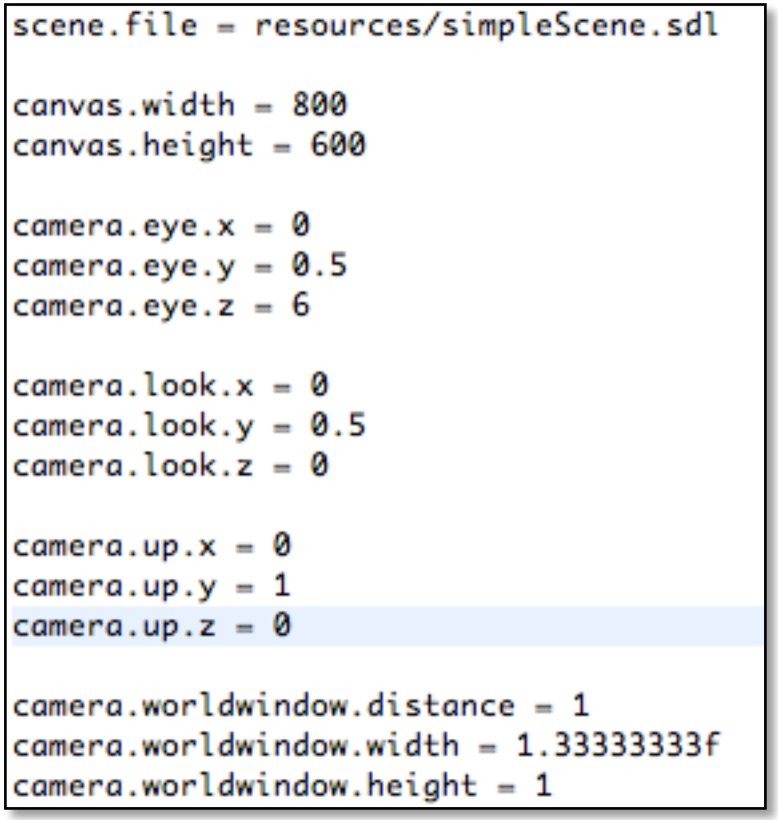
Configuration file

Old configuration file



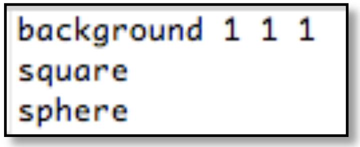
```
scene.file = resources/wineglass.txt

camera.eye.x = 0
camera.eye.y = 0.5
camera.eye.z = 6

camera.look.x = 0
camera.look.y = 0.5
camera.look.z = 0

camera.up.x = 0
camera.up.y = 1
camera.up.z = 0
```

New configuration file



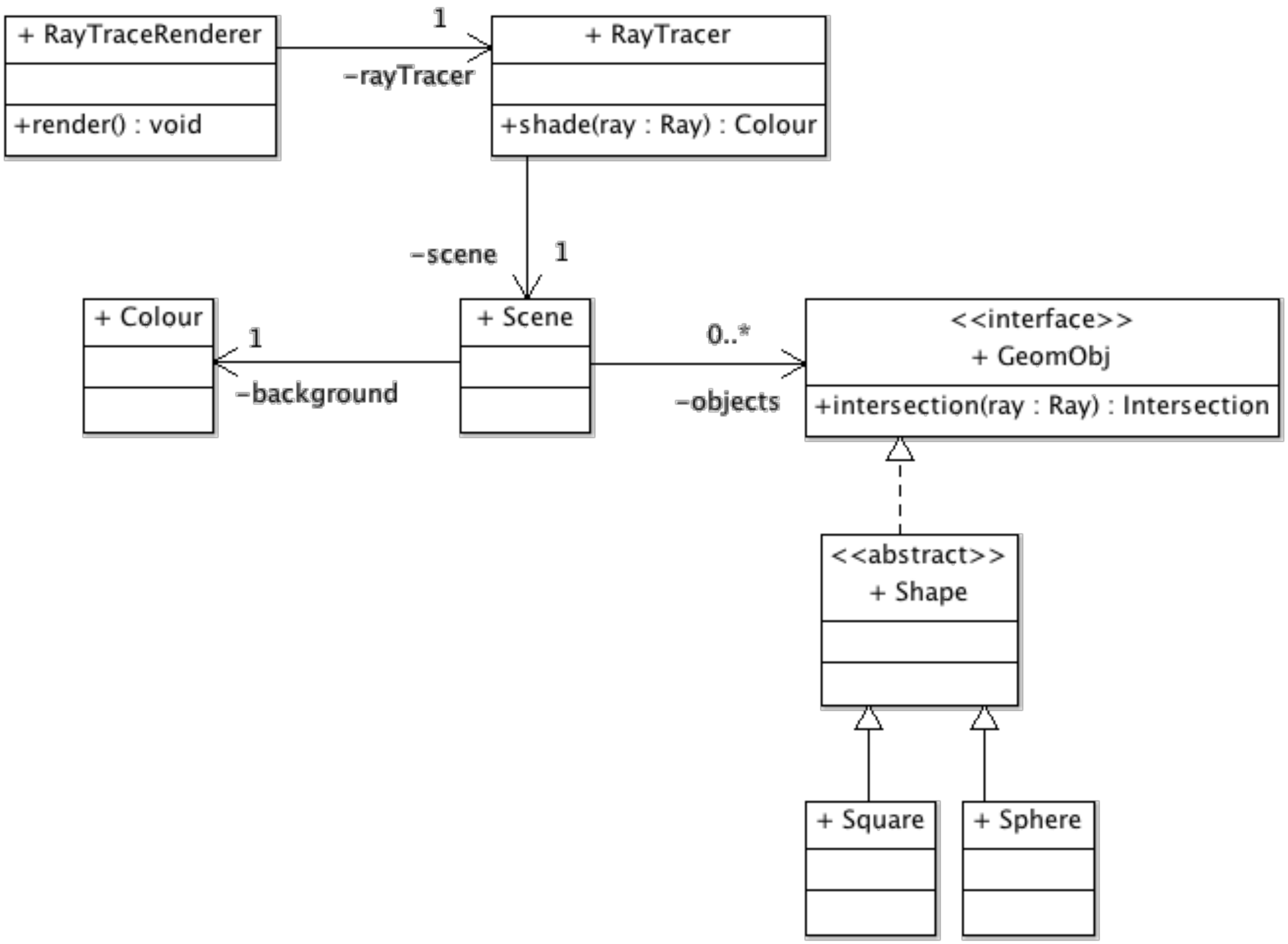
```
scene.file = resources/simpleScene.sdl

canvas.width = 800
canvas.height = 600

camera.eye.x = 0
camera.eye.y = 0.5
camera.eye.z = 6

camera.look.x = 0
camera.look.y = 0.5
camera.look.z = 0

camera.up.x = 0
camera.up.y = 1
camera.up.z = 0

camera.worldwindow.distance = 1
camera.worldwindow.width = 1.33333333f
camera.worldwindow.height = 1
```

simpleScene.sdl



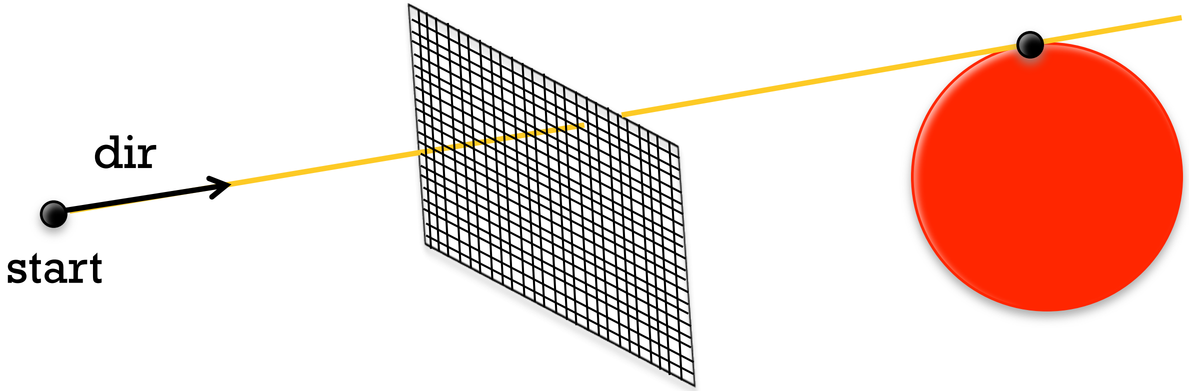
```
background 1 1 1
square
sphere
```

(Simplified) rendering framework



Intersection ray – generic sphere

A generic sphere is a sphere centered around the origin and with radius one.



For every point P on the ray, there exists a positive real number t such that

$$P = start + t.dir$$

The t values which satisfy the equation

$$(dir.dir)\ t^2 + (2.start.dir)\ t + start.start - 1 = 0$$

correspond to the hitpoint(s).

Intersection ray – generic sphere

$$(\text{dir.dir})\, t^2 + (2.\text{start.dir})\, t + \text{start.start} - 1 = 0$$

is of the form

$$at^2 + bt + c = 0$$

which is a quadratic equation which can be solved by computing the discriminant D.

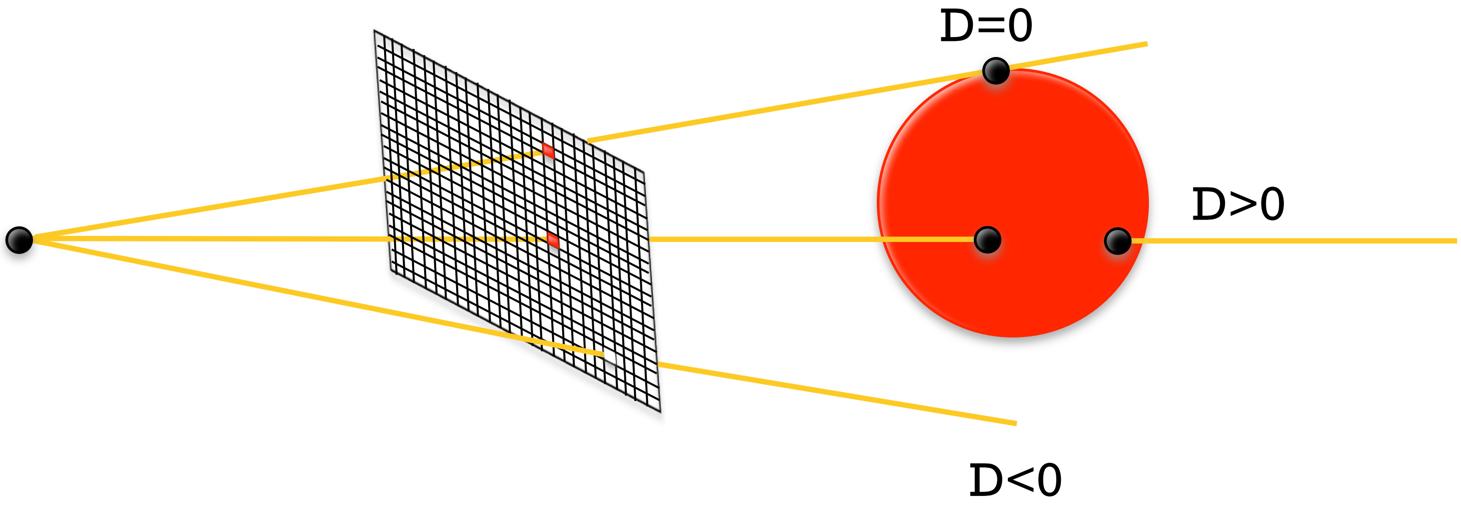
$$D = b^2 - 4ac \qquad t = \frac{-b \pm \sqrt{D}}{2a}$$

$D<0 \longrightarrow$ no solution

$D=0 \longrightarrow$ 1 solution

$D>0 \longrightarrow$ 2 solutions

Intersection ray – generic sphere



Intersection ray – generic sphere

$$(\text{dir.dir})\, t^2 + (2.\text{start.dir})\, t + \text{start.start} - 1 = 0$$

is of the form

$$at^2 + bt + c = 0$$

which is a quadratic equation which can be solved by computing the discriminant D.

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

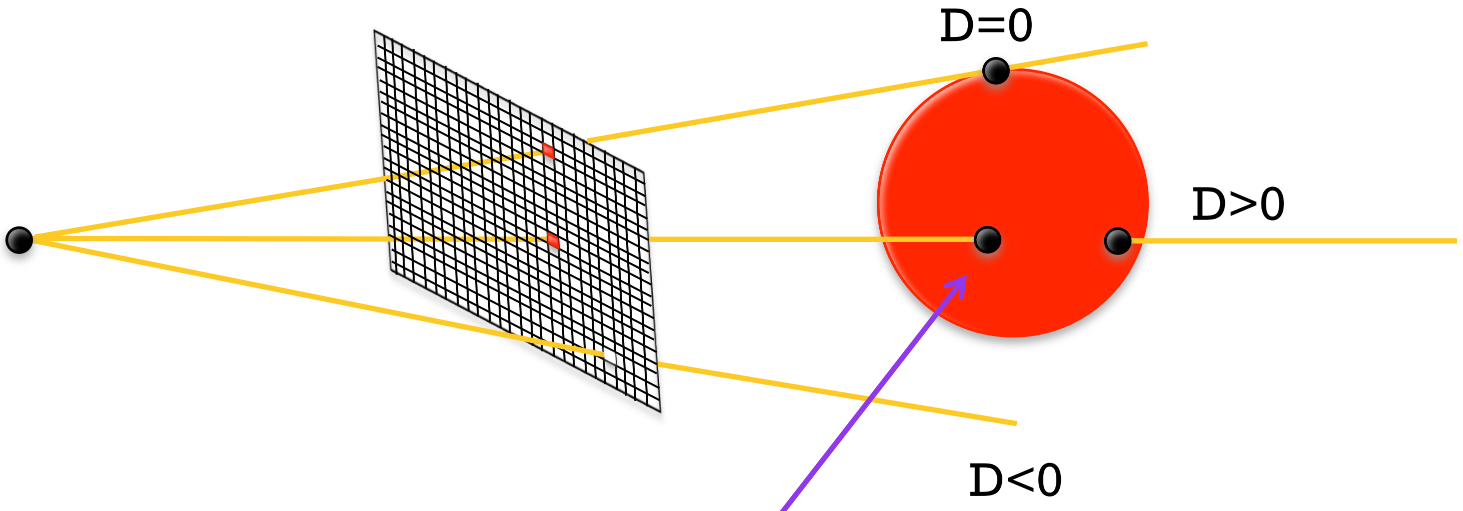
D<0 ⟶ no solution (The ray does not intersect the sphere.)

D=0 ⟶ 1 solution (The ray touches the sphere – one hit)

D>0 ⟶ 2 solutions (There are two hits.)

Which one is relevant for us?

Intersection ray – generic sphere



We are interested in the first hitpoint.
This is the hitpoint closest to the eye of the camera.

Intersection ray – generic sphere

$$(\text{dir.dir})\, t^2 + (2.\text{start.dir})\, t + \text{start.start} - 1 = 0$$

Examples

- Assume this equation has two solutions $t_1 = 1$ and $t_2 = 2$. Which t value corresponds to the hitpoint relevant for us?

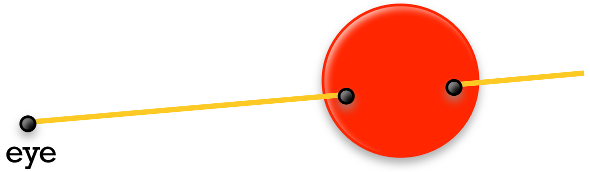
$$t_1 = 1$$

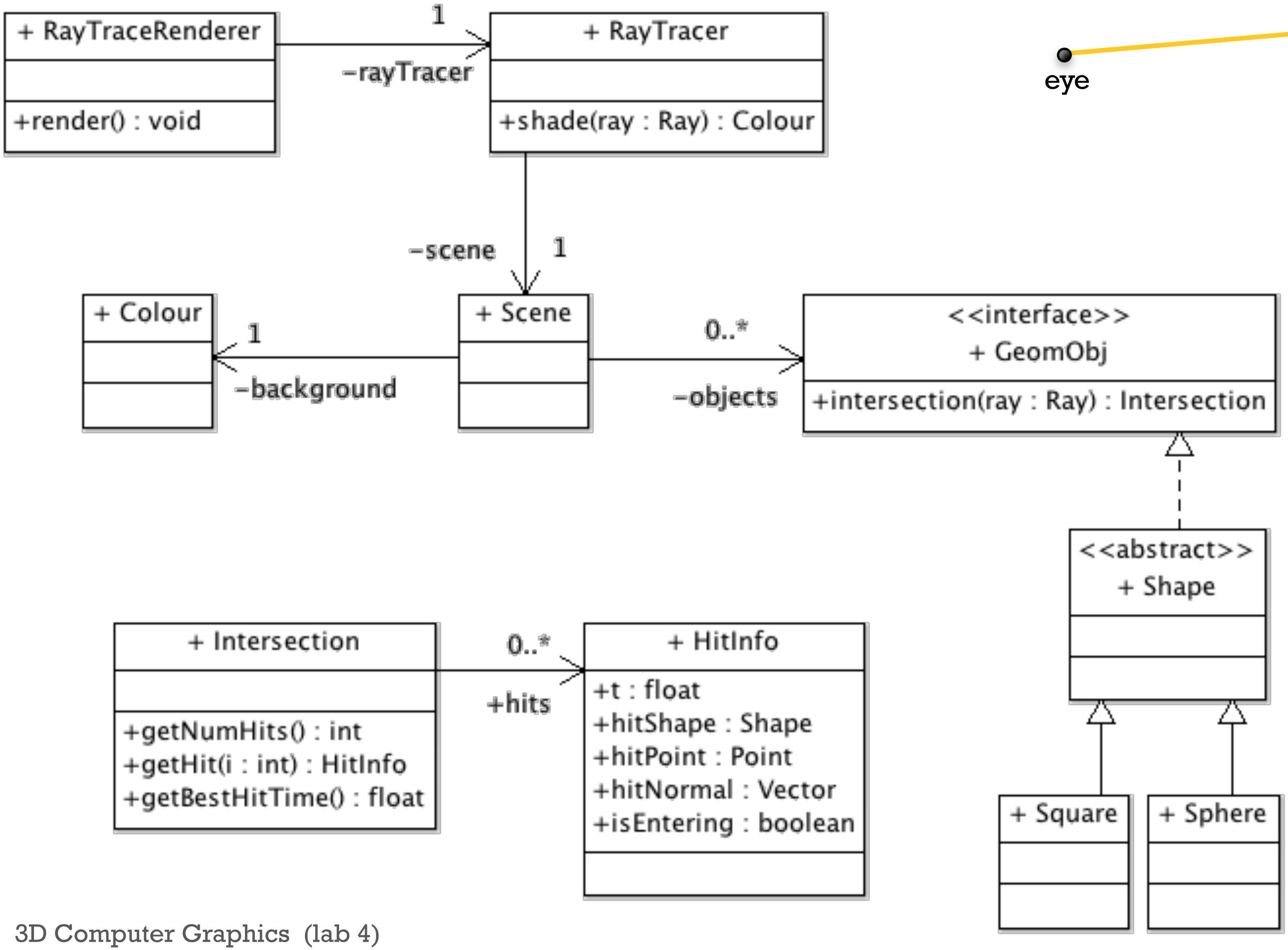
- Assume this equation has two solutions $t_1 = -1$ and $t_2 = 3$. Which t value corresponds to the hitpoint relevant for us?

$$t_2 = 3$$

We are not interested in negative t values as they correspond to hitpoints lying behind the camera!

(Simplified) rendering framework





+ RayTraceRenderer

+render() : void

1

-rayTracer

+ RayTracer

+shade(ray : Ray) : Colour

-scene

1

+ Colour

1

-background

+ Scene

0..*

-objects

<<interface>>
+ GeomObj

+intersection(ray : Ray) : Intersection

<<abstract>>
+ Shape

+ Intersection

+getNumHits() : int
+getHit(i : int) : HitInfo
+getBestHitTime() : float

0..*

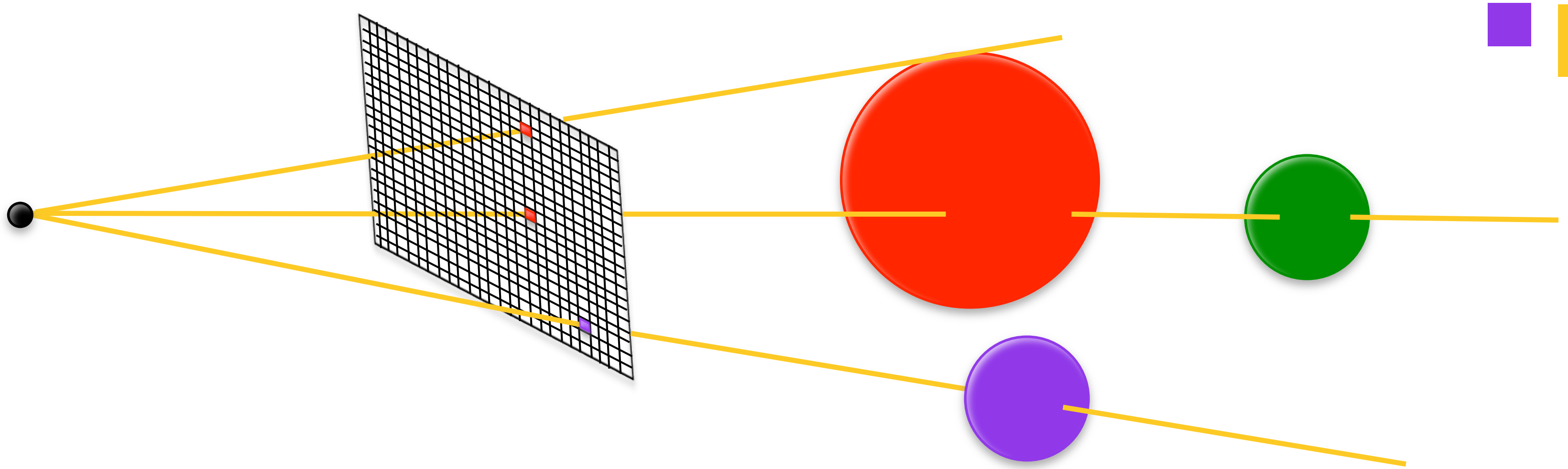
+hits

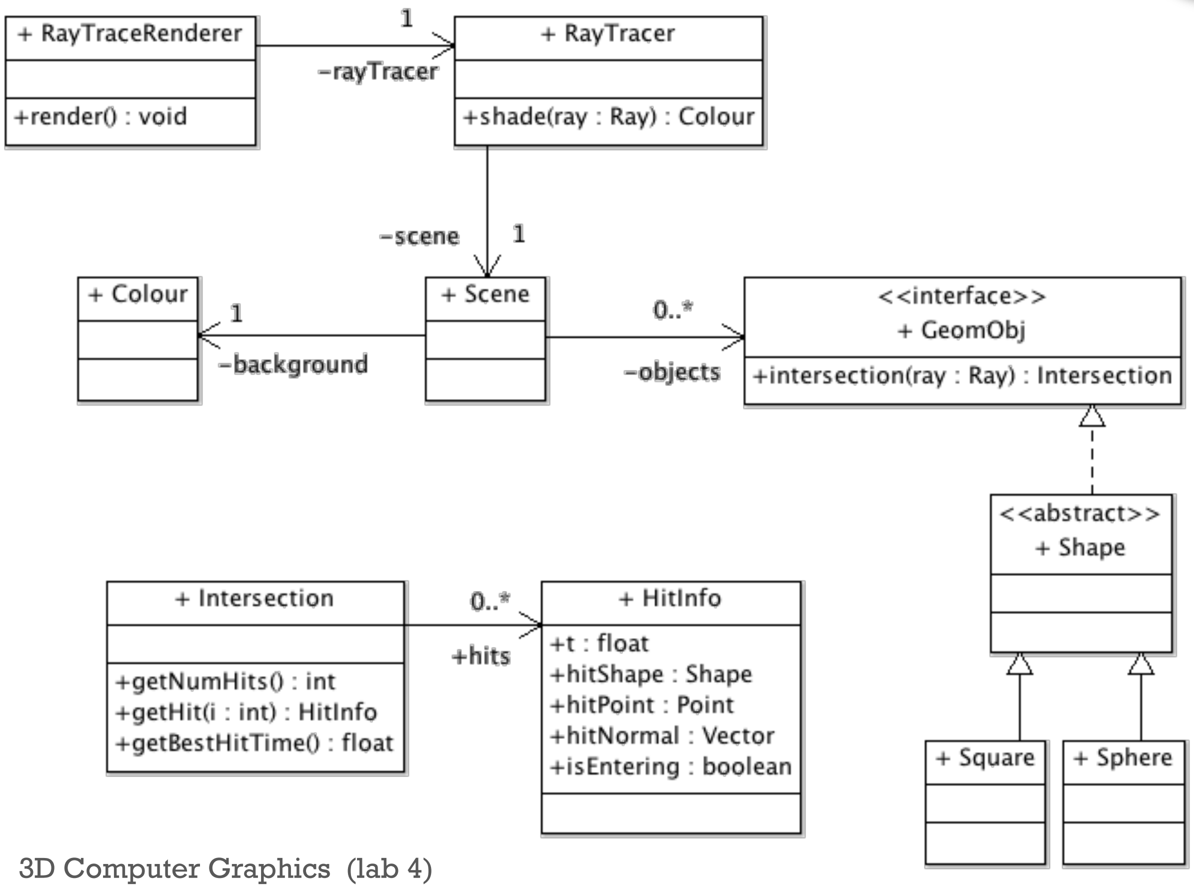
+ HitInfo

+t : float
+hitShape : Shape
+hitPoint : Point
+hitNormal : Vector
+isEntering : boolean

+ Square

+ Sphere





+ RayTraceRenderer

+render() : void

1

-rayTracer

+ RayTracer

+shade(ray : Ray) : Colour

-scene

1

+ Colour

1

-background

+ Scene

0..*

-objects

<<interface>>
+ GeomObj

+intersection(ray : Ray) : Intersection

<<abstract>>
+ Shape

+ Square

+ Sphere

+ Intersection

+getNumHits() : int
+getHit(i : int) : HitInfo
+getBestHitTime() : float

0..*

+hits

+ HitInfo

+t : float
+hitShape : Shape
+hitPoint : Point
+hitNormal : Vector
+isEntering : boolean

shade(ray : Ray) : Colour

```
For each object in the scene:
        Compute the intersection
        with the ray

Determine the best intersection,
this is the one with the lowest
bestHitTime.

If this best Intersection object has
no hits,
return the background colour.
Else return a red colour
```

Questions?