



DH2323 DGI17

COMPUTER GRAPHICS AND INTERACTION

INTRODUCTION TO **RAYTRACING**

Christopher Peters

CST, KTH Royal Institute of Technology,
Sweden

chpeters@kth.se

<http://kth.academia.edu/ChristopherEdwardPeters>

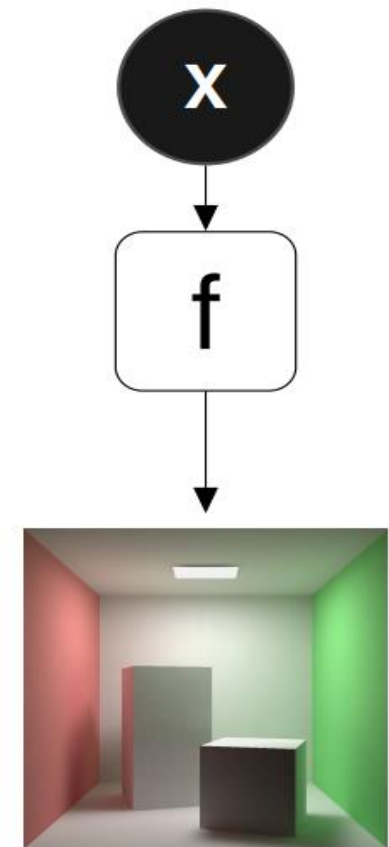
Image Synthesis

In computer graphics, create images based on a *model*

Recall:

An underlying process generates observations

Can control generation through parameters



Nice Results

"Pebbles" by Jonathan Hunt



"Bonsai Life" by Jeremy M. Praay



"Glasses" by Gilles Tran

Nice Results

"Distant Shores" by Christoph Gerber



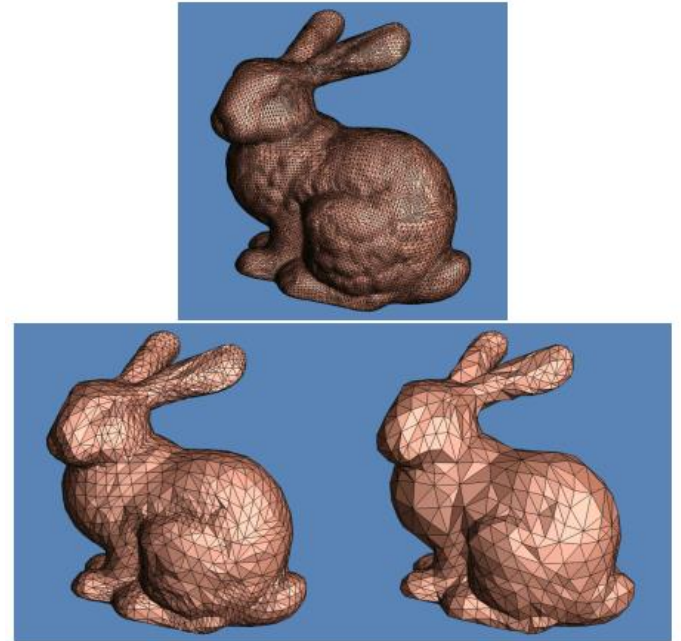
"Christmas Baubles" by Jaime Vives Piqueres



"Still with Bolts" by Jaime Vives Piqueres

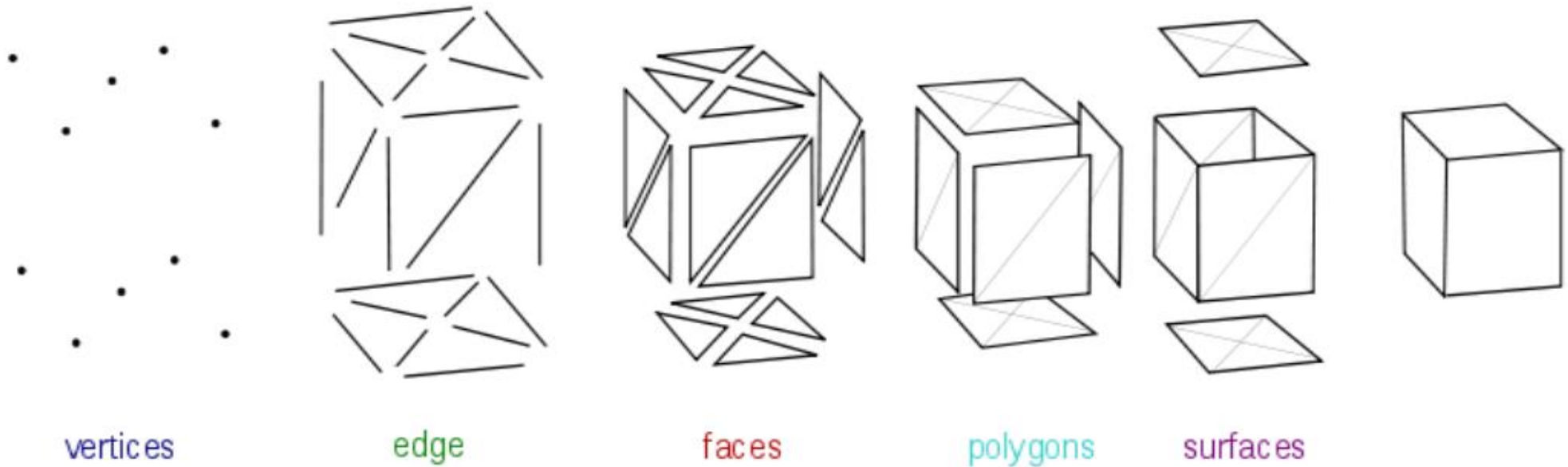
Some Constituents I

- Geometry



Geometric primitives

(a brief introduction)

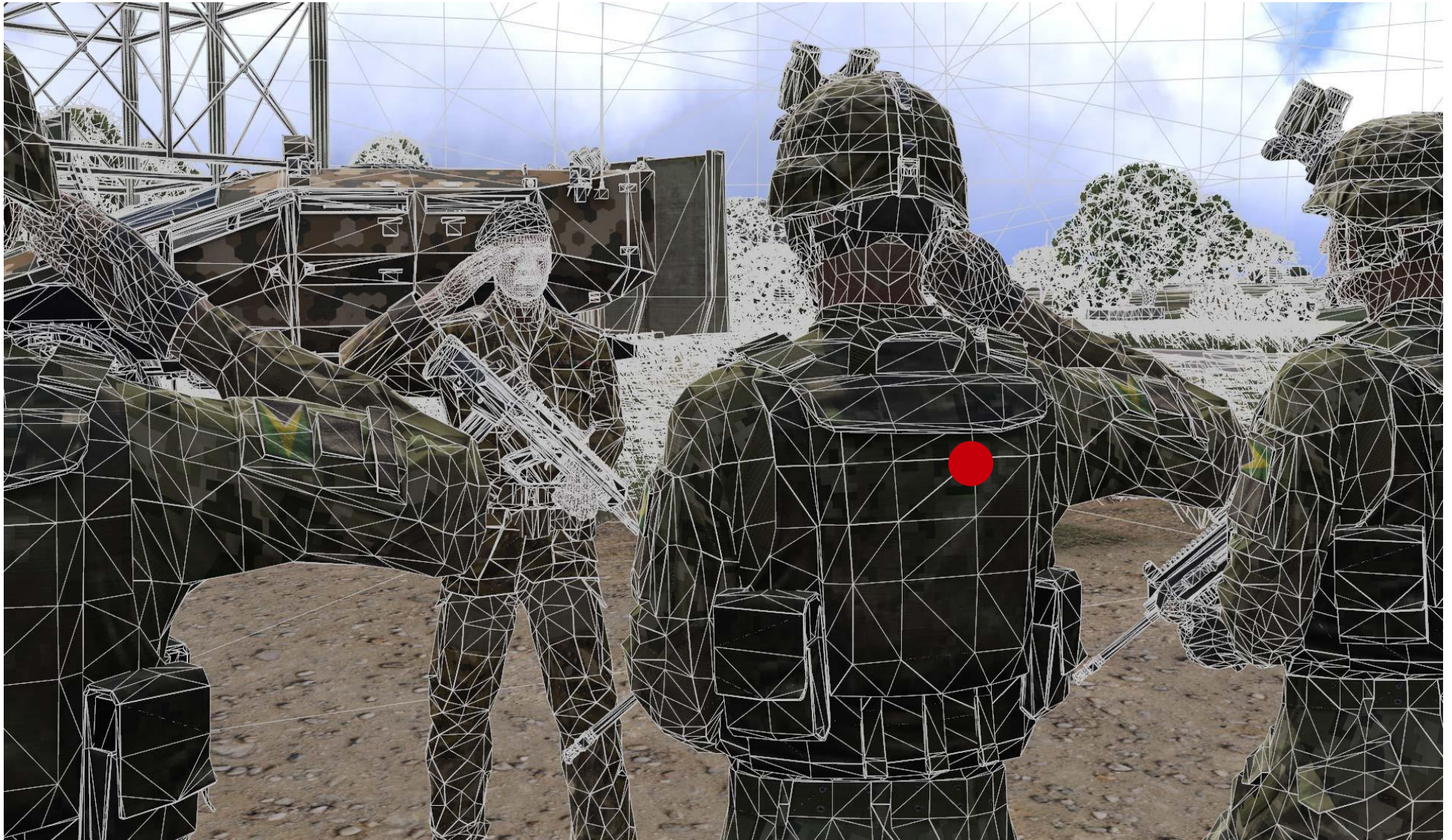


Triangle meshes



Arma3, Bohemia Interactive

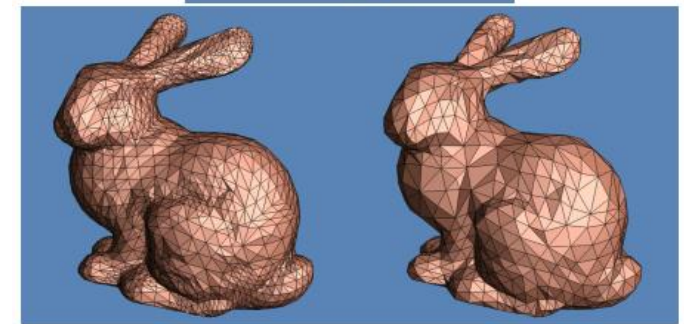
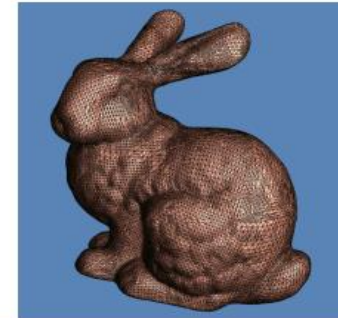
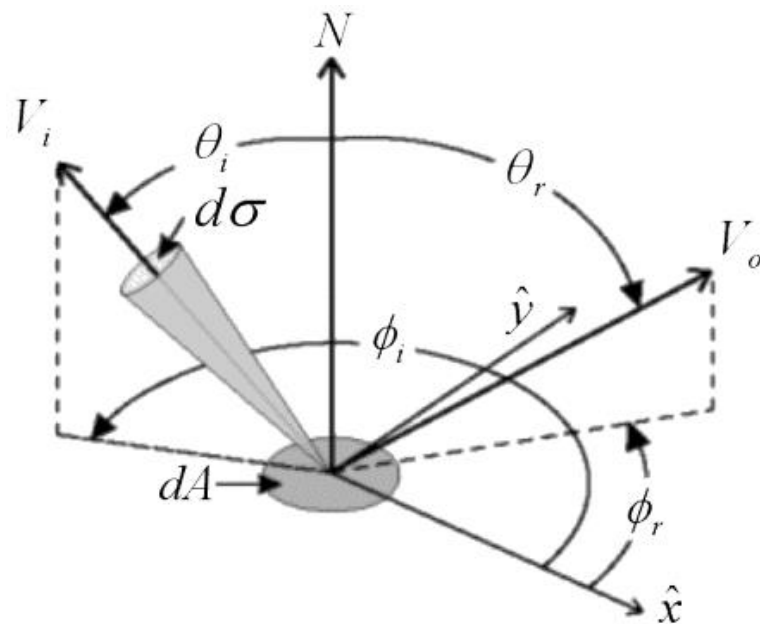
Point clouds



Arma3, Bohemia Interactive

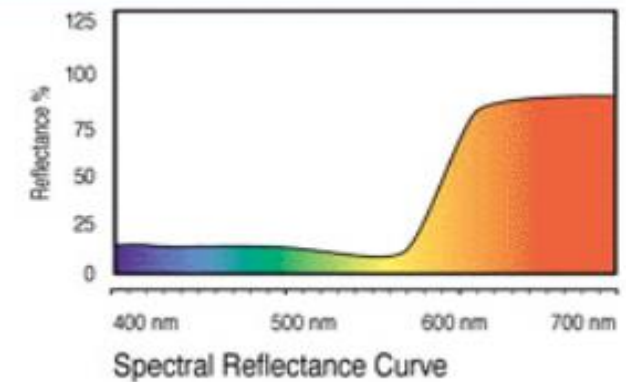
Some Constituents I

- Geometry
- **Lighting**
- Surface properties
- Anything else?



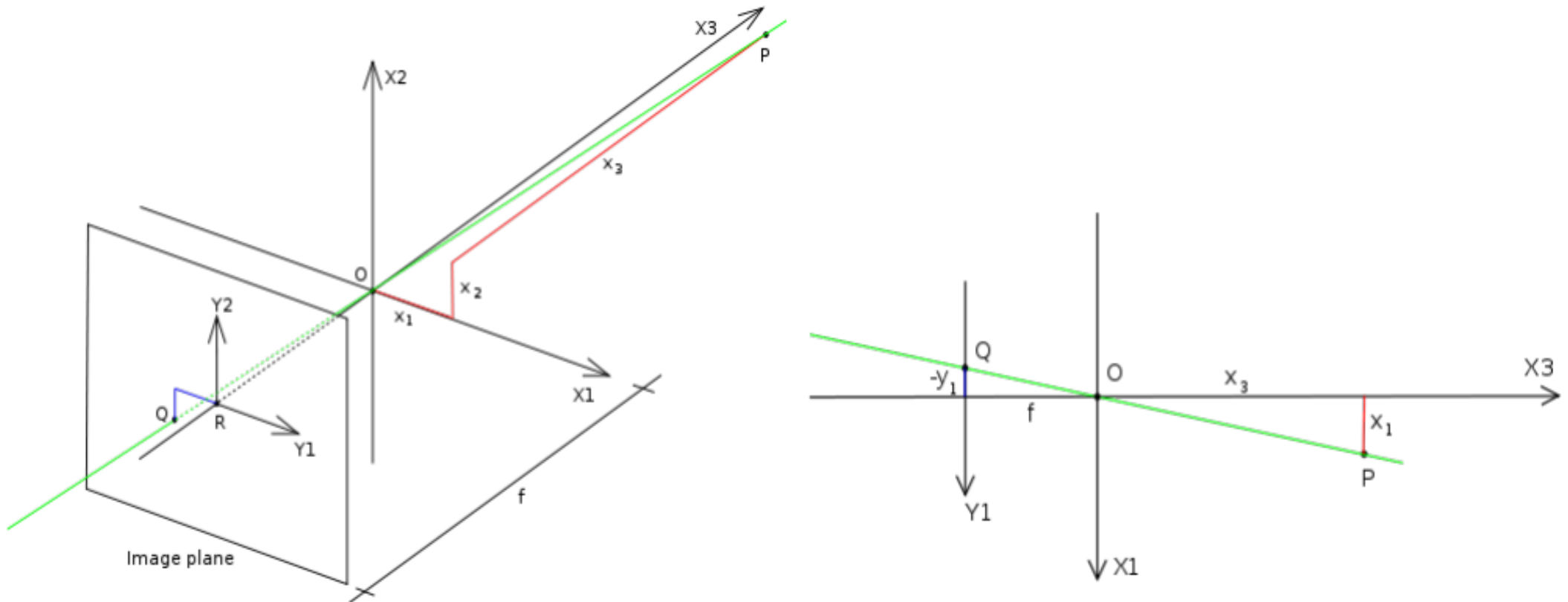
Red
Object

=



A Camera!

- Camera Model (pinhole)



Master of your own (virtual) Universe

- You determine the physical laws (and models) that define your virtual world and simulate it
- How is the transportation of light simulated?
- Does gravity affect the environment?
- Can solid objects, such as people, move through each other / the environment?
- Should a ball bounce when it hits the ground and how should it deform?
- How do fluids act? In what cases do objects float?
- How do nebulae disperse and galaxies interact?

Master of your own (virtual) Universe



I-Novae Engine, I-Novae Studios

Master of your own (virtual) Universe

- You determine the physical laws (and models) that define your virtual world and simulate it
- How is the transportation of light simulated?
- Does gravity affect the environment?
- Can solid objects, such as people, move through each other / the environment?
- Should a ball bounce when it hits the ground and how should it deform?
- How do fluids act? In what cases do objects float?
- How do nebulae disperse and galaxies interact?
- But there's a catch...

The catch

- These laws do not exist in the virtual world by default
- Must computationally **model** and **program** them

Idea 1

Use the concept of *light rays* for modelling transport of light

Define light sources that emit rays

Test for intersections between rays and geometric shapes in the scene

When a ray hits the surface of a shape

- See how much light energy bounces i.e. is *reflected*

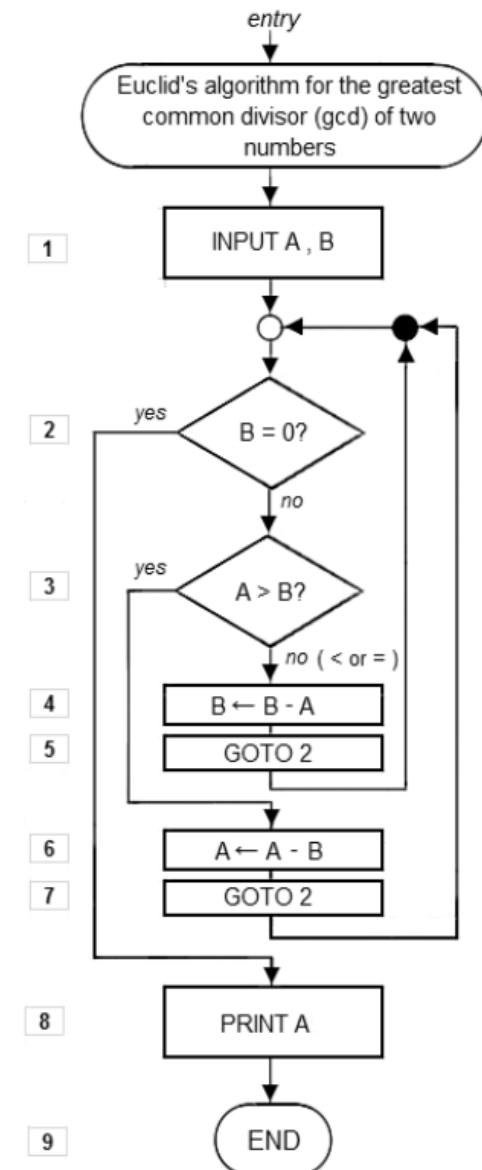
When a ray hits the image plane of the camera

- Render a colour to the screen

Question

How do we go from this *idea* to a computational model?

i.e. how to define an algorithm or step-by-step procedure



Answer: Remember this?

Interactive computer graphics is essentially:

Mathematics programming

Quite possibly the most fun and rewarding maths programming you will ever do*

**disclaimer: you'll get from it what you put in*

Maths Programming Toolkit

- Define a toolkit of mathematical operations
- Each will be important for our final algorithm
- Important keywords from idea 1:
 - Light rays from light sources
 - Intersections with geometry
 - Bounces/reflects off surfaces
 - Render to screen

What is a 'Light Ray'?

- Concept
 - Idealised narrow beam of light (optics)
 - Discrete, particles
- Geometrically speaking:
 - Similar in some ways to a straight line
 - Has a starting point and direction
 - But extends infinitely in defined direction
- Mathematically:

$$\mathbf{r}_0 = [x_0, y_0, z_0]^T$$

$$\mathbf{r}_d = [x_d, y_d, z_d]^T, \quad ||\mathbf{r}_d|| = 1$$

$$\mathbf{r}_t = \mathbf{r}_0 + t \cdot \mathbf{r}_d$$

One degree-of-freedom

Where do they come from?

- Emitted from light sources
- Parameterised
 - Position
 - Colour
 - Intensity

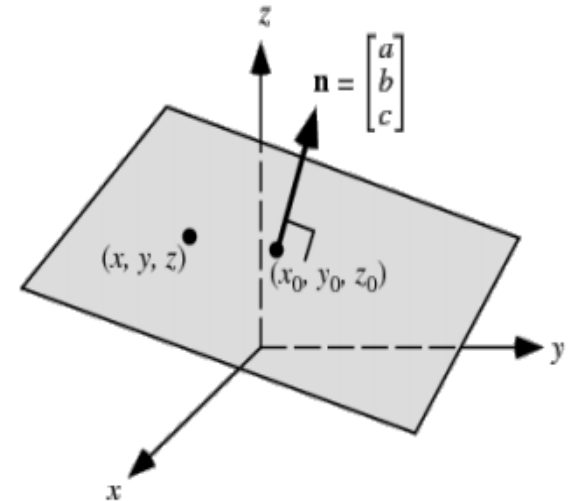
Intersections

- Need to be able to test for intersections between rays and scene geometry (objects)
- Different types of geometry:
 - Planes
 - Spheres
 - Triangles
- Resolving intersections involve solving equations

Ray-plane Intersection

- Plane defined as:

- Plane normal $\mathbf{n} = [a, b, c]$
- Unit normal $\|\mathbf{n}\|_2 = 1$
- d offset to origin
- Equation $a \cdot x + b \cdot y + c \cdot z + d = 0$
- Two degrees-of-freedom*



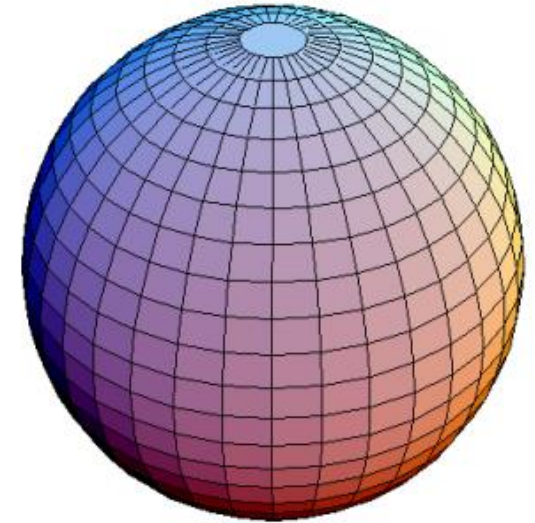
- Intersection point:

$$\mathbf{p}_i = \mathbf{r}_0 - \frac{\mathbf{n}^T \mathbf{r}_0 + d}{\mathbf{n}^T \mathbf{r}_d} \cdot \mathbf{r}_d$$

Ray-sphere Intersection

- Sphere defined as:

- Center of sphere $\mathbf{x}_c = [x_c, y_c, z_c]^T$
- Radius r
- $(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = r^2$



- Intersections:

$$\mathbf{p}_i = -\mathbf{r}_d^T(\mathbf{r}_0 - \mathbf{r}_c) \pm \sqrt{\mathbf{r}_d^T(\mathbf{r}_0 - \mathbf{r})_c - (\mathbf{r}_0 - \mathbf{x}_c)^T(\mathbf{r}_0 - \mathbf{x}_c) + r^2}$$

Worked Example

(recommend to work this out later using a pen and paper)

Find the intersections, if any, between the Ray with $r_0 = (0,2,0)^T$, $r_d = (0,-1,0)^T$ and the Sphere with $x_c = (0,0,0)^T$, $r = 1$

Apply the quadratic formula $t = -b \pm \sqrt{b^2 - 4ac} / (2a)$ to find two solutions, where:

$$a = r_d \cdot r_d$$

$$b = 2r_d \cdot (r_0 - x_c)$$

$$c = (r_0 - x_c) \cdot (r_0 - x_c) - r^2$$

The value of $b^2 - 4ac$ indicates how many roots the equation has, where negative number indicates no intersections between the ray and sphere, a zero indicates a single intersection on the edge of the sphere and a positive number indicates two intersections where the ray enters and exits the sphere. In this example, $b^2 - 4ac$ is positive indicating **two intersections**.

$$\text{Apply formula; } r_d \cdot r_d t^2 + 2r_d \cdot (r_0 - x_c)t + (r_0 - x_c) \cdot (r_0 - x_c) - r^2 = 0$$

Entering the above value gives **$t^2 - 4t + 3 = 0$**

$$\Rightarrow \mathbf{t = 3 \text{ and } t = 1}$$

Recalling ray equation: $r_0 + t \cdot r_d$

$$t=1: (0,2,0) + (0,-1,0) = \mathbf{(0,1,0)} \quad \dots \text{ first intersection point}$$

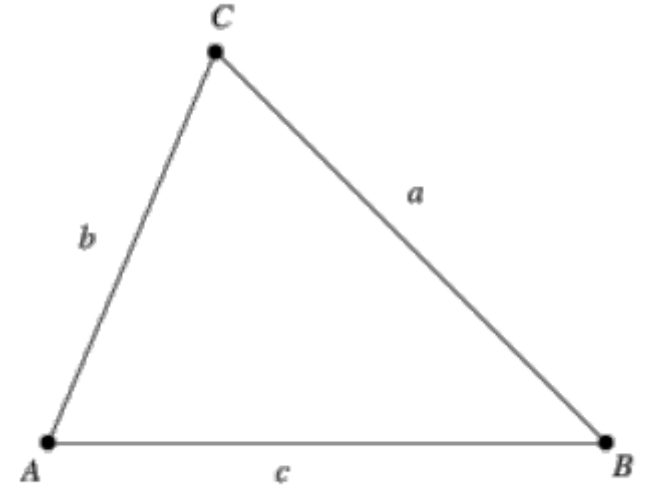
$$t=3: (0,2,0) + (0,-3,0) = \mathbf{(0,-1,0)} \quad \dots \text{ second intersection point}$$

Ray-triangle Intersection

- Triangle defined as:

- Three vertices

- ▶ $\mathbf{t}_i = [x, y, z]^T, i = 1 \dots 3$



Ray-triangle Intersection

- Triangle defined as:

- Three vertices

- $\mathbf{t}_i = [x, y, z]^T, i = 1 \dots 3$

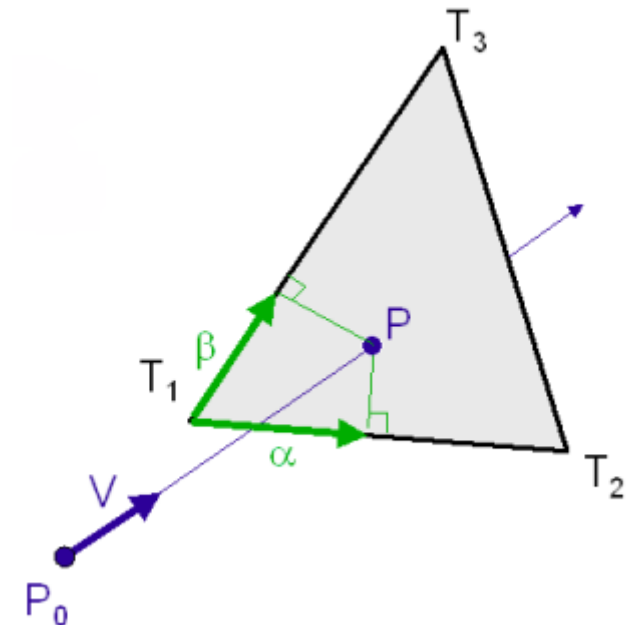
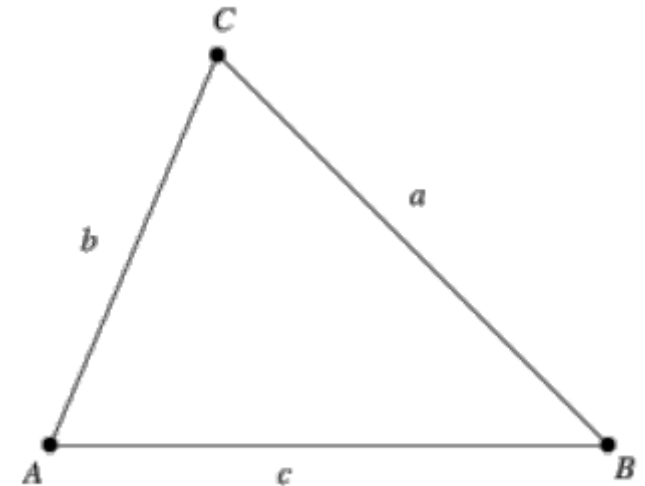
- Intersection:

1. Check collision with plane

2. Check if inside triangle

- $0 \leq \alpha, \beta \leq 1$

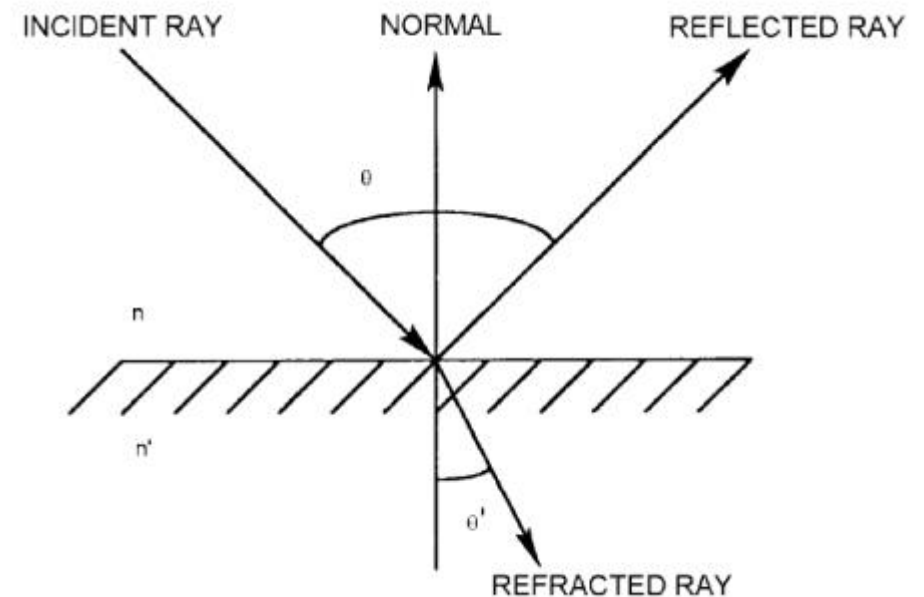
- $\alpha + \beta \leq 1$



Surfaces

Must also consider rays hitting and bouncing off surfaces

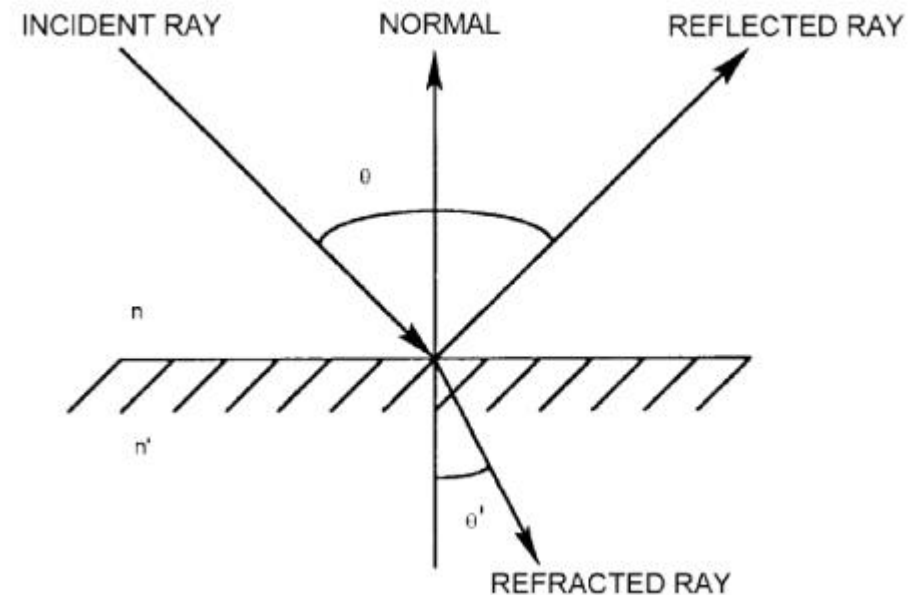
1. Incoming ray hits surface
2. Divided into,
 - ▶ reflected component
 - ▶ refracted component



Surfaces

Must also consider rays hitting and bouncing off surfaces

1. Incoming ray hits surface
2. Divided into,
 - ▶ reflected component
 - ▶ refracted component

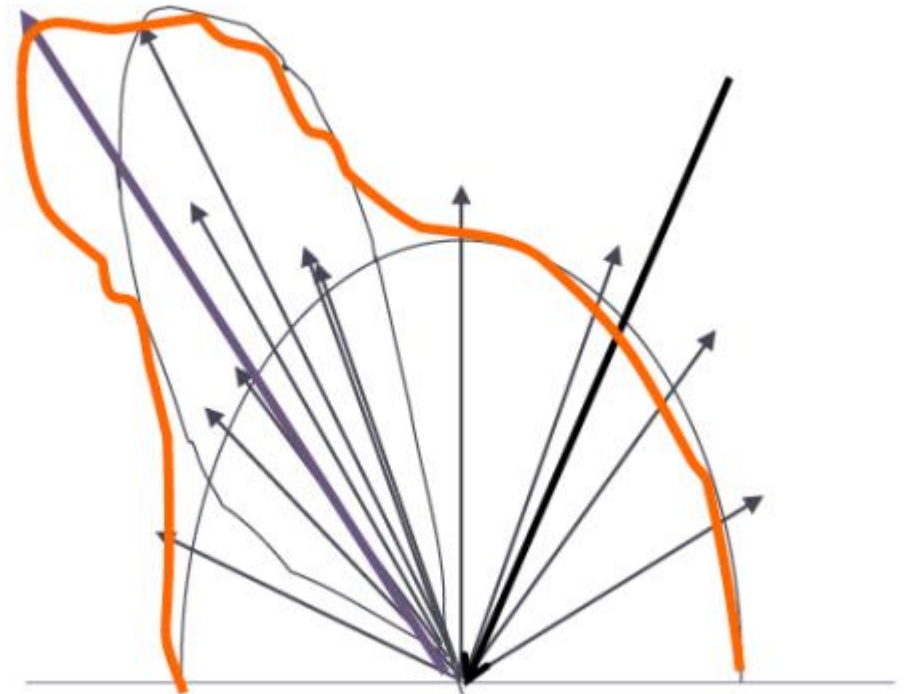


Surface types can reflect/refract rays in different ways

Accurate Reflection

Complex opaque surfaces scatter incoming light in many different directions

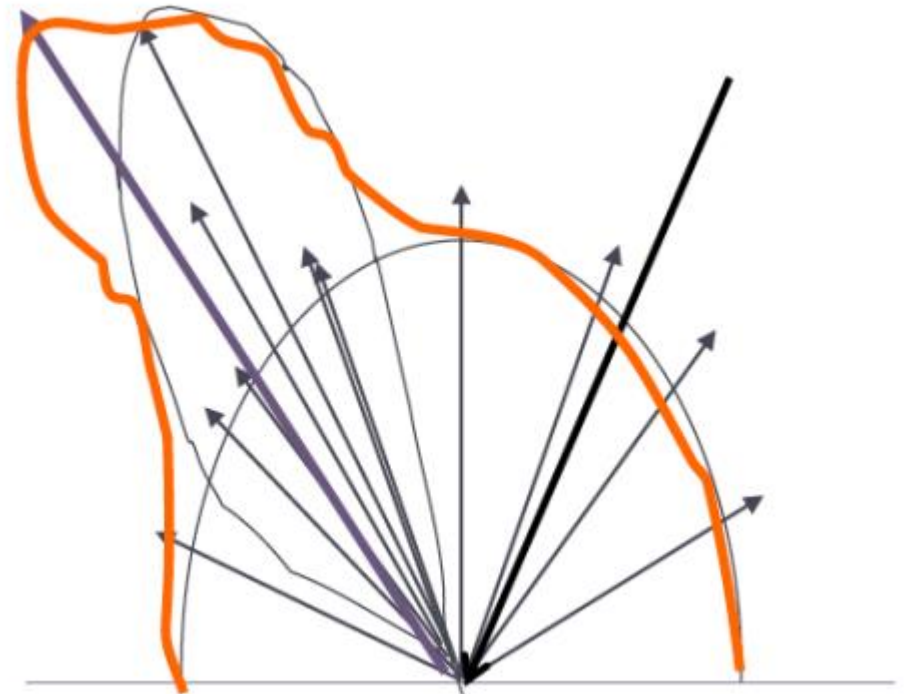
- BRDF - **B**idirectional **R**eflectance **D**istribution **F**unction



Accurate Reflection

Complex opaque surfaces scatter incoming light in many different directions

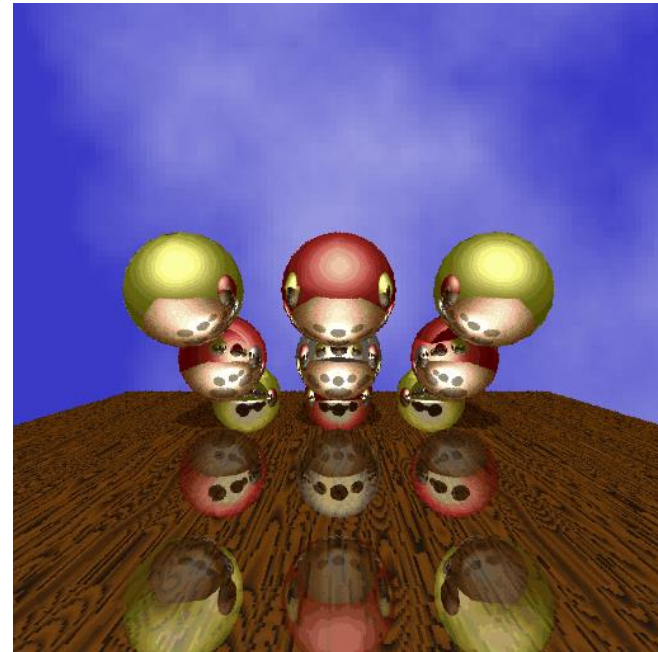
- BRDF - **B**idirectional **R**eflectance **D**istribution **F**unction



Where does this information come from?

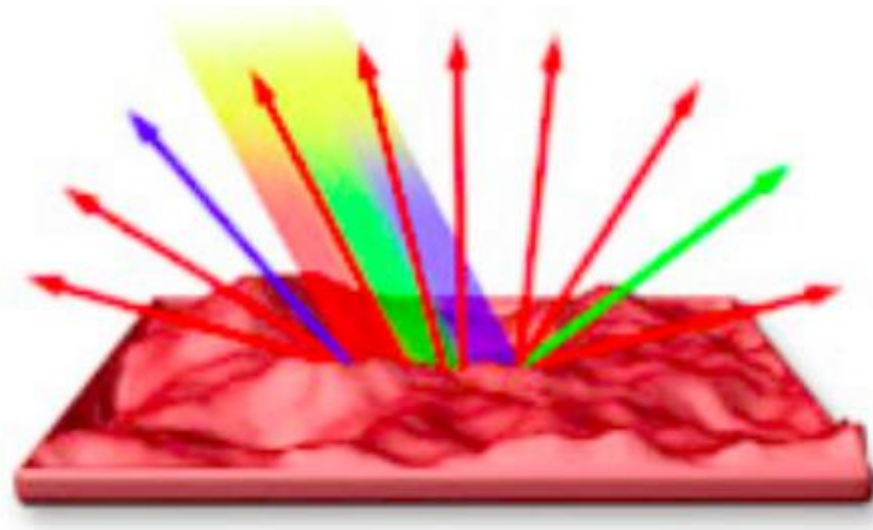
Surfaces

Specular surfaces are like mirrors
Light from a single incoming direction
reflected into a single outgoing direction



Surfaces

Diffuse (matte) surfaces are more rough
Light from a single incoming direction
reflected in multitude of outgoing directions
Lambertian surface

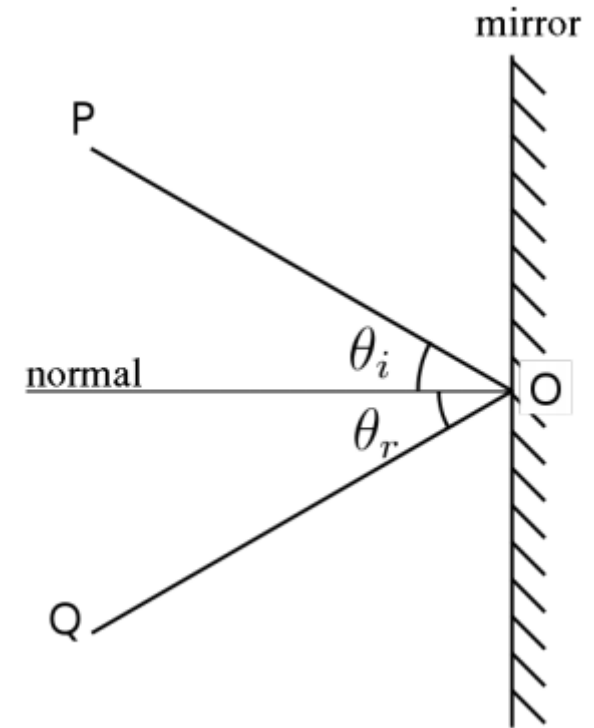


Modelling Specular Surfaces

Incoming ray P , outgoing ray Q and face normal are in the same plane

Angle from normal the same between incoming and outgoing ray

Rays P and Q are on opposite sides of the face normal



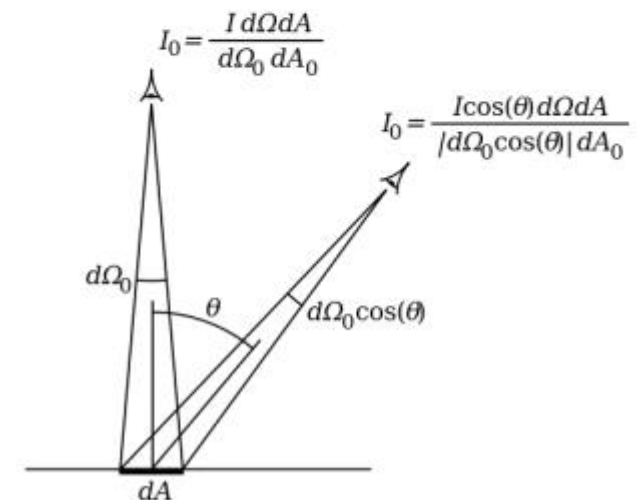
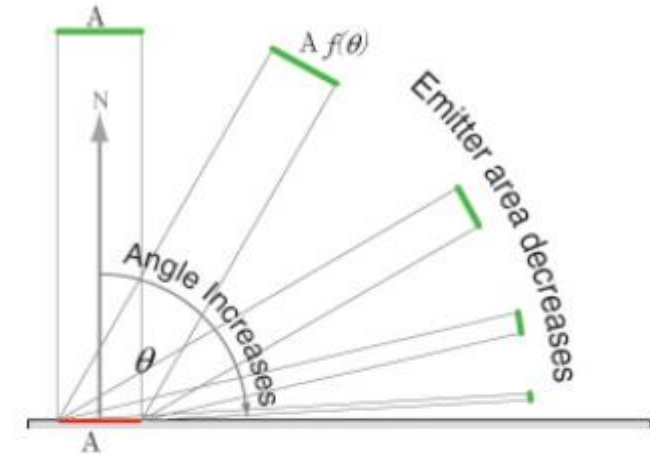
$$\mathbf{d}_o = \mathbf{d}_i - 2(\mathbf{d}_i^T \cdot \mathbf{n})\mathbf{n}$$

Modelling Diffuse Surfaces

Lambertian reflection

Surface is equally bright independent of viewing angle

Isotropic – uniform in all directions



Lambert's Cosine Law

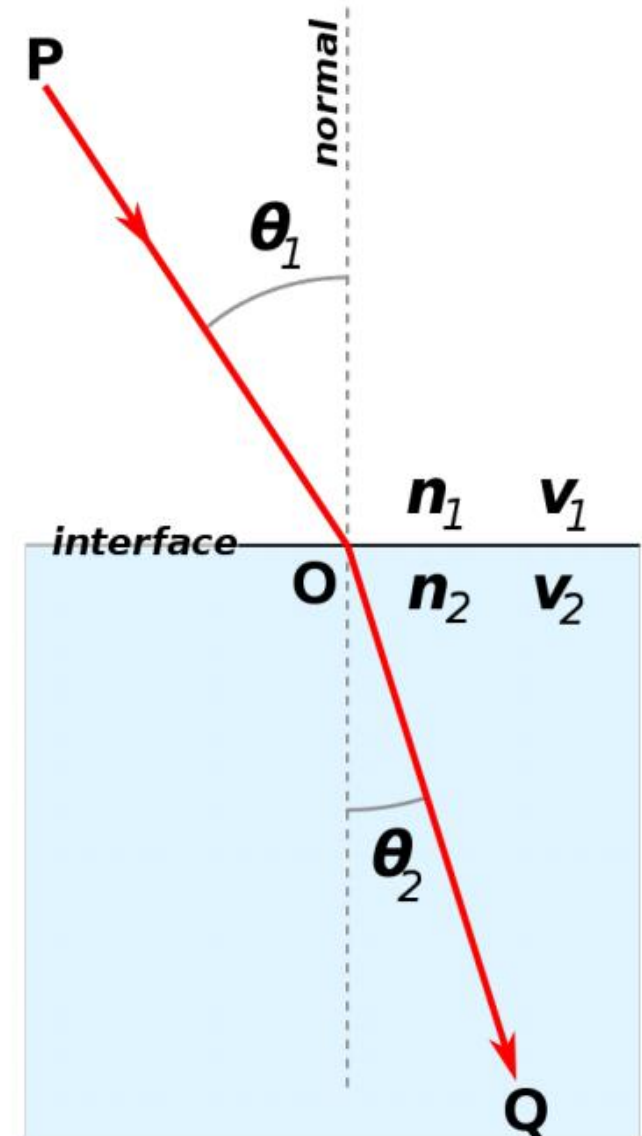
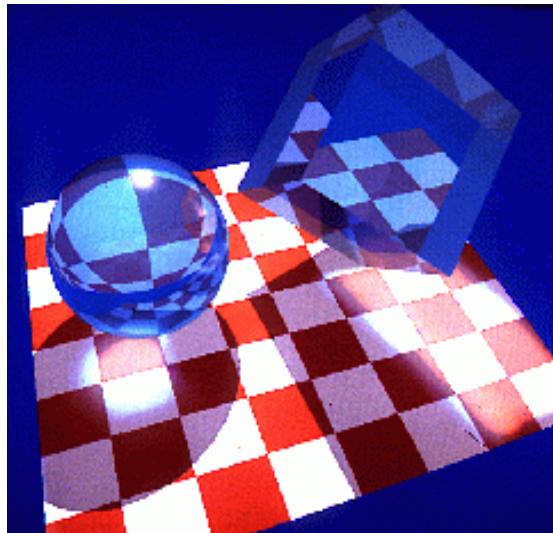
$$I_0 = I \frac{\cos(\theta) d\Omega dA}{\cos(\theta) d\Omega_0 dA_0}$$

Modelling Refraction

- Snell's Law

$$\frac{\sin(\theta_1)}{\sin(\theta_2)} = \frac{v_1}{v_2} = \frac{n_2}{n_1}$$

- ▶ Refractive Index of Material



Some Light Types

Ambient Lighting

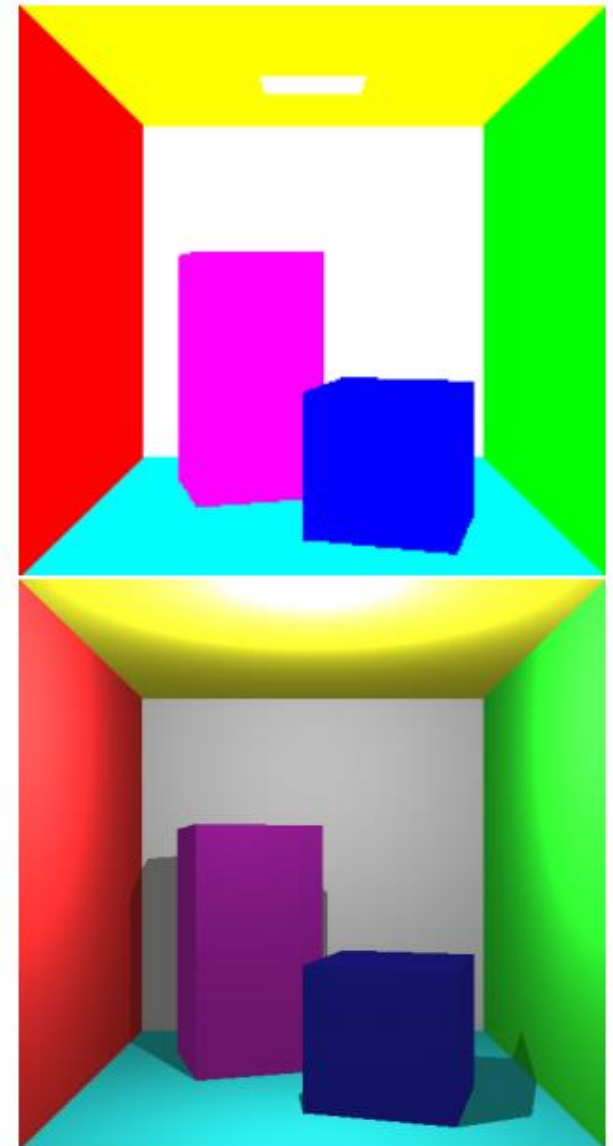
Constant light affecting each part of the scene equally

Point Light Sources

Rays travelling from a point in all directions

Spotlight Sources

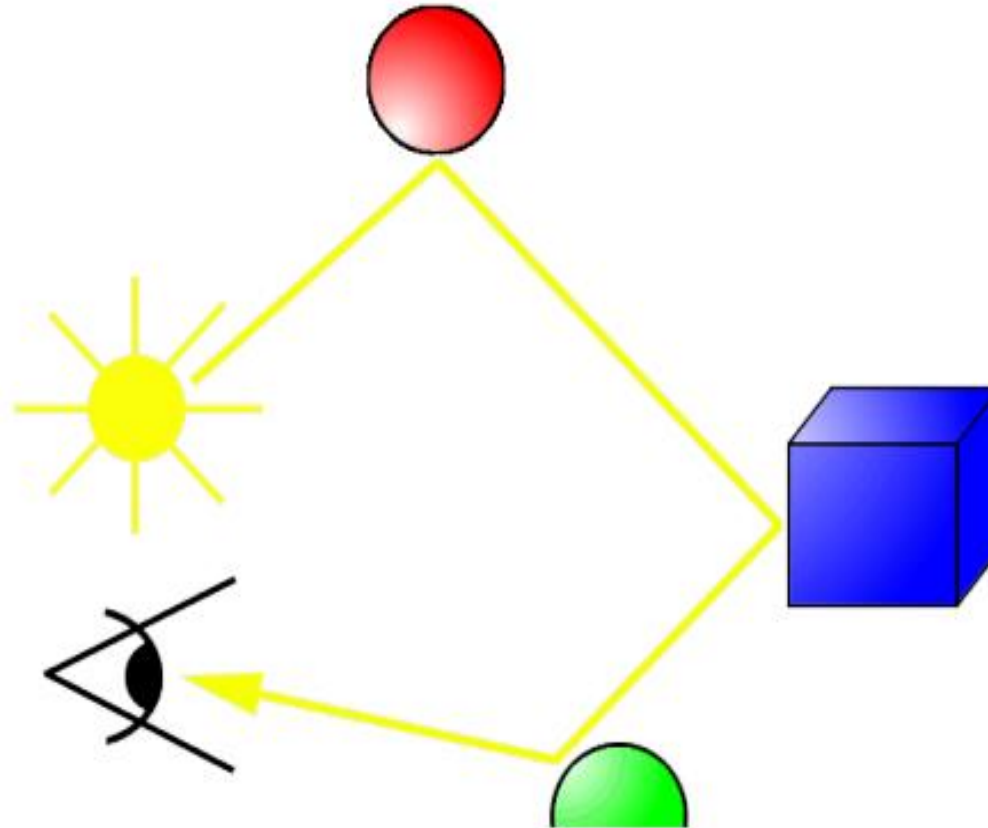
Rays travelling from a point in limited directions



Our Initial Idea

We have now defined some modelling elements

Remember our initial idea:



Question

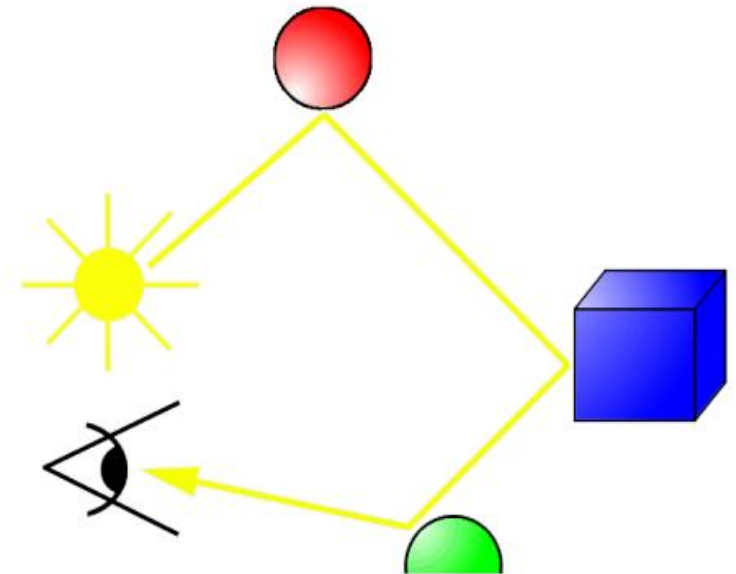
Is this feasible??

Question

Is this feasible??

Think of what the procedure may look like:

```
for(i=0;i<nr_rays,i++)  
{  
  while(!hit image&&!infinity)  
  {  
    /*  
    compute new direction  
    compute new colour  
    */  
  }  
}
```



Idea 2

Cast rays of light backwards through the viewing plane, into the scene

Test for intersections between each ray and virtual objects in the scene

Idea 2

Cast rays of light backwards through the viewing plane, into the scene

Test for intersections between each ray and virtual objects in the scene

Ray fails to hit anything

- Background or default colour returned

Idea 2

Cast rays of light backwards through the viewing plane, into the scene

Test for intersections between each ray and virtual objects in the scene

Ray fails to hit anything

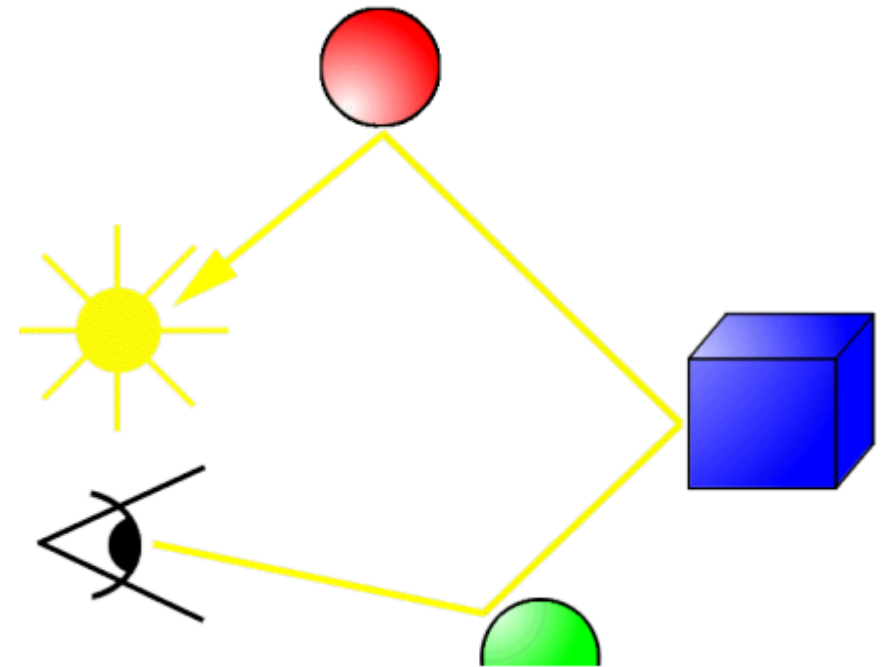
- Background or default colour returned

When ray hits (i.e. 'intersects') first object

- Cast ray from intersection point on object to light sources
- Test material properties of object

Backwards Raytracing

```
for(v=0;v<height,v++)  
{  
  for(u=0;u<width;v++)  
  {  
    for(i=0;i<max_nr_bounce;i←  
      ++)  
    {  
      /*  
      compute new direction  
      compute new colour  
      */  
    }  
  }  
}
```



Termination Criteria

1. No intersection

Termination Criteria

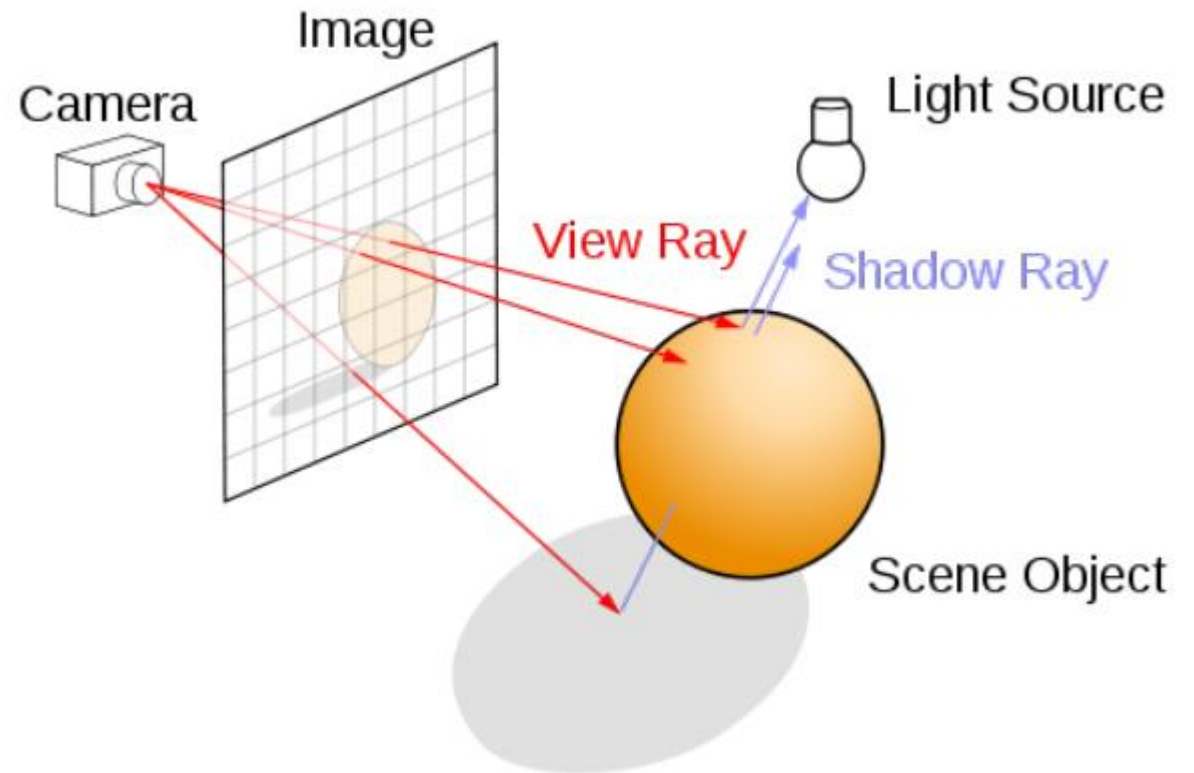
1. No intersection
2. Reach maximal depth
Number of bounces

Termination Criteria

1. No intersection
2. Reach maximal depth
Number of bounces
3. Contribution of secondary ray attenuated
below threshold
Each reflection/refraction bounce attenuates
ray

Shadows

1. Shoot ray
2. Intersection
 - ▶ Shoot ray to light
 - ▶ Free/Blocked



Raytracing in General



Quake Wars: Ray-traced

Simple algorithm

Capable of simulating
complex light interactions

Well-suited for software
rendering

Very flexible

Easy to incorporate new
effects

Not always viewed as so
interactive

But things are changing

Overall

One of a number of *global illumination* models

Global illumination

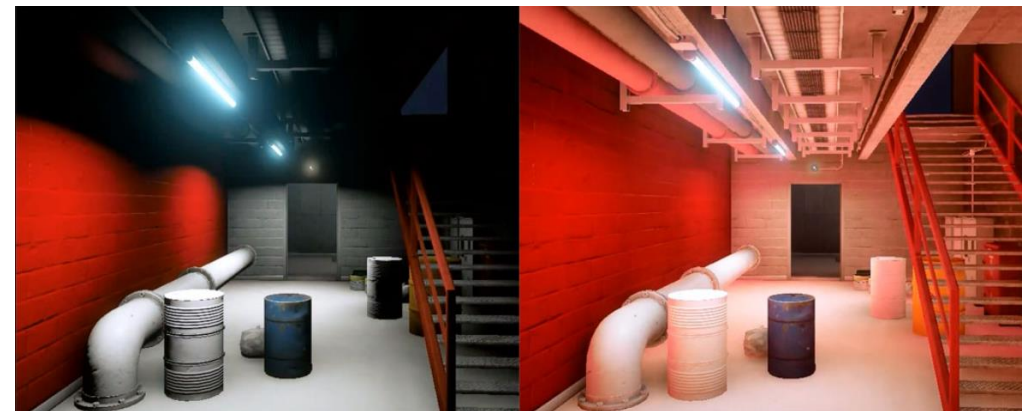
- More realism
- More computation

Others include:

1. *Radiosity*
2. *Photon mapping*
3. *Path tracing*
4. *Ambient occlusion*



Museum scene with radiosity



Battlefield 3 radiosity

Next lecture

- More details about projects
- Next Monday (3rd April)
- 10:00 – 12:00 D2
- Labs:
 - Download and start to look at the first lab
 - You are doing well at this stage if you have a basic version building
 - The C++ tutorial and introduction lab session may run again (according to demand)