Computer Graphics

P. Healy

CS1-08
Computer Science Bldg.
tel: 202727
patrick.healy@ul.ie

Spring 2021–2022

Outline

Introduction

- A global illumination-based rendering method
- Powerful technique that is basis of many photorealistic rendering algorithms
- Based on idea of **ray casting** which was used to identify visible surfaces in a scene by "firing off" a ray from a pixel and seeing what is the closest surface it lands on
- This idea is now generalised to let ray "bounce around" the scene to collect intensity contributions

Introduction

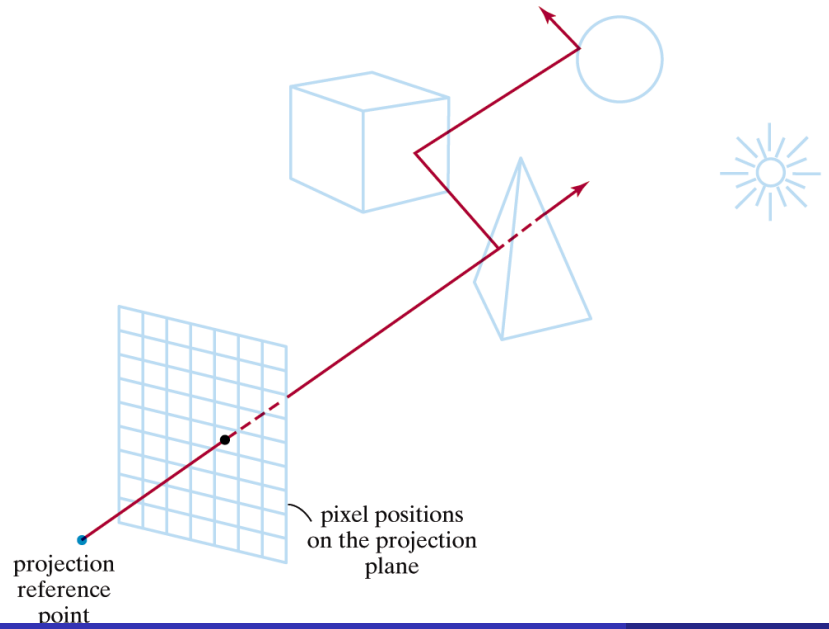
- A global illumination-based rendering method
- Powerful technique that is basis of many photorealistic rendering algorithms
- Based on idea of **ray casting** which was used to identify visible surfaces in a scene by "firing off" a ray from a pixel and seeing what is the closest surface it lands on
- This idea is now generalised to let ray "bounce around" the scene to collect intensity contributions

Introduction

- A global illumination-based rendering method
- Powerful technique that is basis of many photorealistic rendering algorithms
- Based on idea of **ray casting** which was used to identify visible surfaces in a scene by “firing off” a ray from a pixel and seeing what is the closest surface it lands on
- This idea is now generalised to let ray “bounce around” the scene to collect intensity contributions

Introduction

- A global illumination-based rendering method
- Powerful technique that is basis of many photorealistic rendering algorithms
- Based on idea of **ray casting** which was used to identify visible surfaces in a scene by “firing off” a ray from a pixel and seeing what is the closest surface it lands on
- This idea is now generalised to let ray “bounce around” the scene to collect intensity contributions



Introduction (contd.)

- Original algorithm was proposed in 1980 although the idea had been proposed previously
- Good news: extremely realistic method of rendering scenes
- Bad news: extremely long computation times
- In addition to being able to detect global reflection effects basic algorithm can be put to use detecting visible surfaces, identifying shadow areas, assisting in rendering transparency effects and handling multiple light sources

Calculating Pixel Intensity

- For each pixel we project a ray from the projection reference point (PRP) **into** the scene and follow it as it bounces off objects, etc.
- Due to the lighting in the scene and, having applied our illumination model, each point will have a light intensity
- The intensity of light at any point will shine along the ray **towards the viewer**
- Therefore, this ray is the reverse of a light ray
- As the ray bounces off objects we record the intensities of the points it hits and we add these to the overall intensity for the **pixel**
- When we generate a pixel-ray the list of surfaces in the scene is processed to determine if the ray intersects any
- The distance to the surface is also computed and this tells us what is visible from that pixel

Calculating Pixel Intensity (contd.)

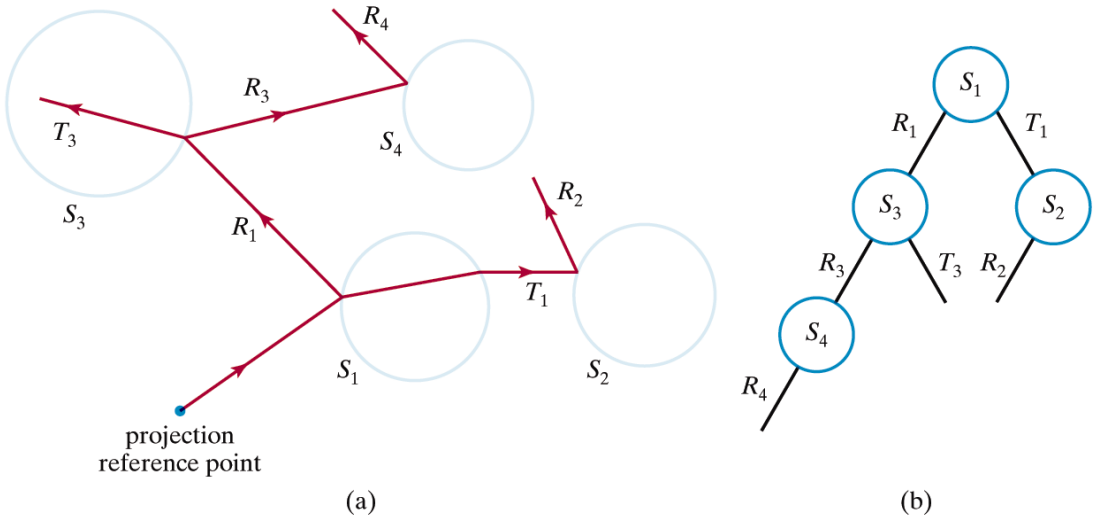
- Once we find the closest surface we now bounce the ray off that surface along a *specular* path (R below)
- In addition, although we haven't discussed transparency effects, when light shines on an object some will be transmitted through the object if it is *transparent* (T below)
- The ray is split in to possibly two parts
- To keep track of these splittings a binary **ray-tracing tree** is used to record the single ray's meanderings

Calculating Pixel Intensity (contd.)

- Once we find the closest surface we now bounce the ray off that surface along a *specular* path (R below)
- In addition, although we haven't discussed transparency effects, when light shines on an object some will be transmitted through the object if it is *transparent* (T below)
- The ray is split in to possibly two parts
- To keep track of these splittings a binary **ray-tracing tree** is used to record the single ray's meanderings

Calculating Pixel Intensity (contd.)

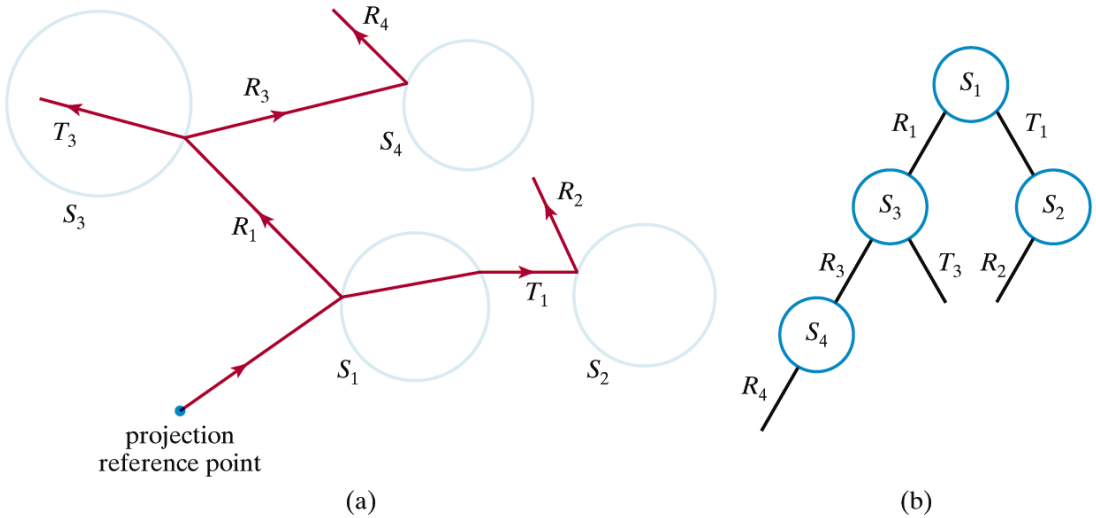
- Once we find the closest surface we now bounce the ray off that surface along a *specular* path (*R* below)
- In addition, although we haven't discussed transparency effects, when light shines on an object some will be transmitted through the object if it is *transparent* (*T* below)
- The ray is split in to possibly two parts



- To keep track of these splittings a binary **ray-tracing tree** is used to record the single ray's meanderings

Calculating Pixel Intensity (contd.)

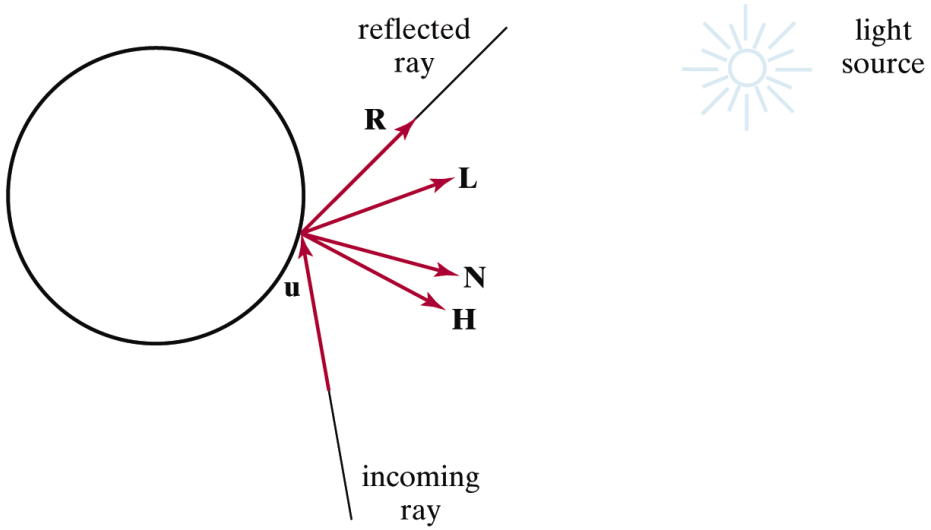
- Once we find the closest surface we now bounce the ray off that surface along a *specular* path (*R* below)
- In addition, although we haven't discussed transparency effects, when light shines on an object some will be transmitted through the object if it is *transparent* (*T* below)
- The ray is split in to possibly two parts



- To keep track of these splittings a binary **ray-tracing tree** is used to record the single ray's meanderings

Calculating Pixel Intensity (contd.)

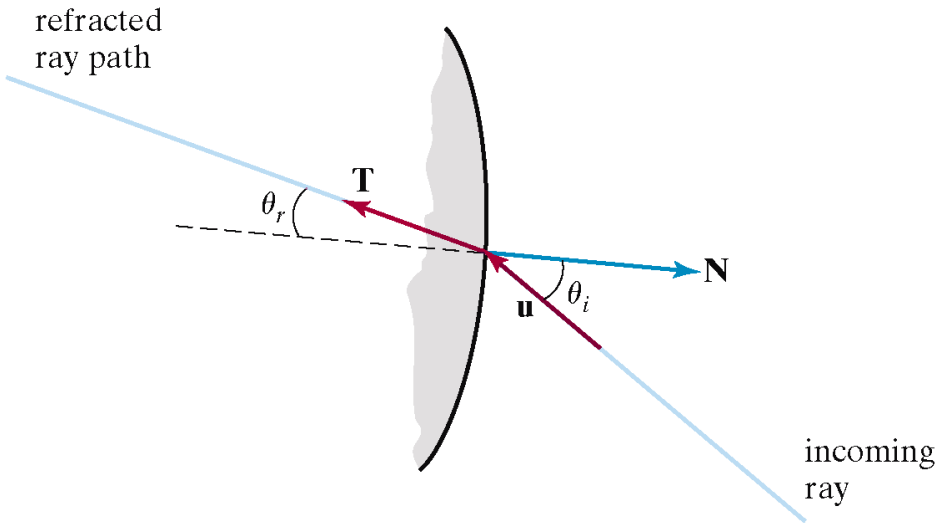
- The rays reflect as follows



As we saw previously $R = u - (2u \cdot N)N$

Calculating Pixel Intensity (contd.)

- The rays refract as follows

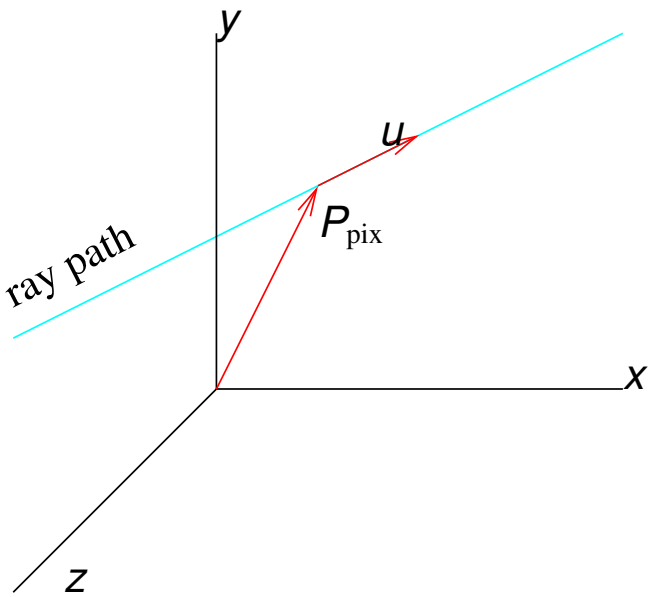


Calculating Pixel Intensity (contd.)

- The binary tree continues growing until
  - The ray intersects no surfaces
  - The ray intersects a light source that **isn't** a reflector
  - The tree has reached maximum allowed depth
- At each surface intersection we apply the illumination model to determine the surface intensity contribution
- Store this at node of RT tree
- When we decide to stop growing the tree we start at leaves and propagate the intensities back up to the next level, taking the distance the light travels in to account (attenuation)
- So, intensity at pixel is the sum of attenuated intensities

Ray-Surface Intersection Calculations

- A ray is described by its initial position and its (unit) direction, **u**



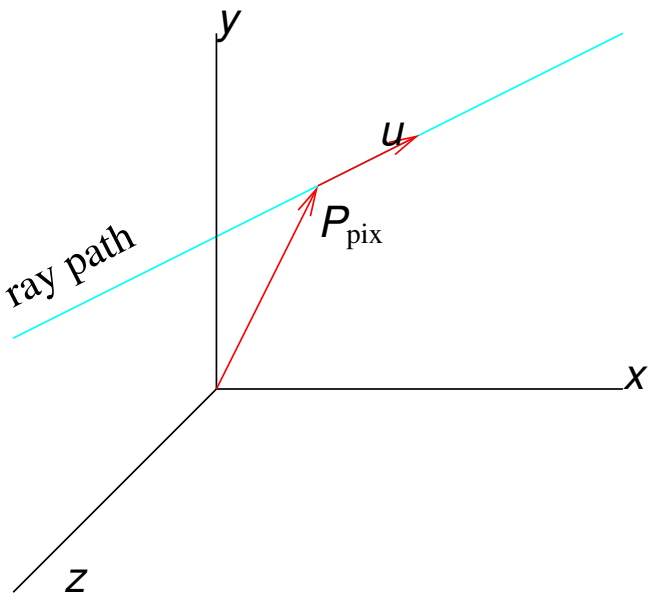
- We will set the ray's initial position to P_{pix} the position of pixel and with ray originating at P_{prp}, the normalized direction vector u is

$$u = \frac{P_{\text{pix}} - P_{\text{prp}}}{|P_{\text{pix}} - P_{\text{prp}}|}$$

- Any point P along ray is then given by

Ray-Surface Intersection Calculations

- A ray is described by its initial position and its (unit) direction, **u**



- We will set the ray's initial position to P_{pix} the position of pixel and with ray originating at P_{prp}, the normalized direction vector u is

$$u = \frac{P_{\text{pix}} - P_{\text{prp}}}{|P_{\text{pix}} - P_{\text{prp}}|}$$

- Any point P along ray is then given by

Ray-Surface Intersection Calculations

- A ray is described by its initial position and its (unit) direction, **u**
- We will set the ray's initial position to P_{pix} the position of pixel and with ray originating at P_{prp}, the normalized direction vector u is

$$u = \frac{P_{\text{pix}} - P_{\text{prp}}}{|P_{\text{pix}} - P_{\text{prp}}|}$$

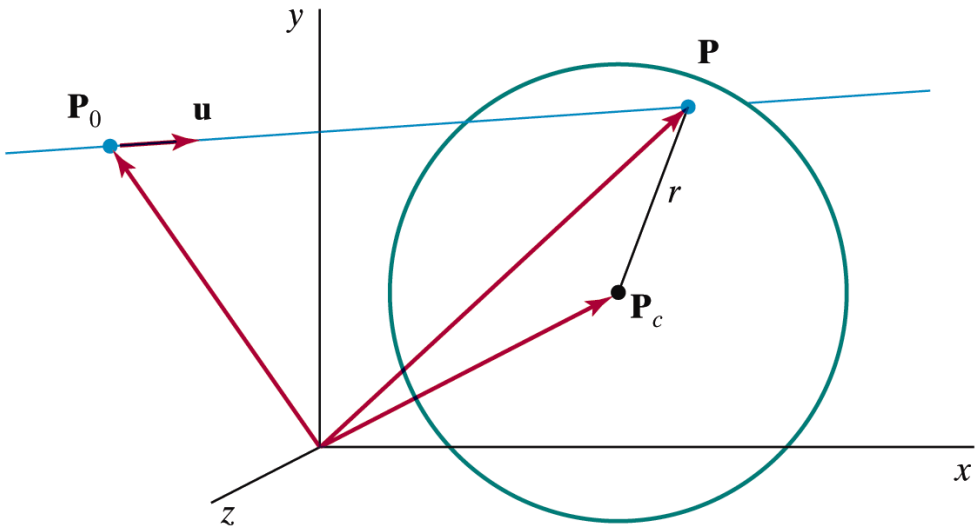
- Any point P along ray is then given by

$$P = P_{\text{pix}} + s\mathbf{u}$$

where s (a scalar) can be used as a measure of the distance to P

Ray-Sphere Intersections

- Given a sphere of radius r and centre P_c and a point P on sphere



- The point P must satisfy

$$|P - P_c|^2 - r^2 = 0$$

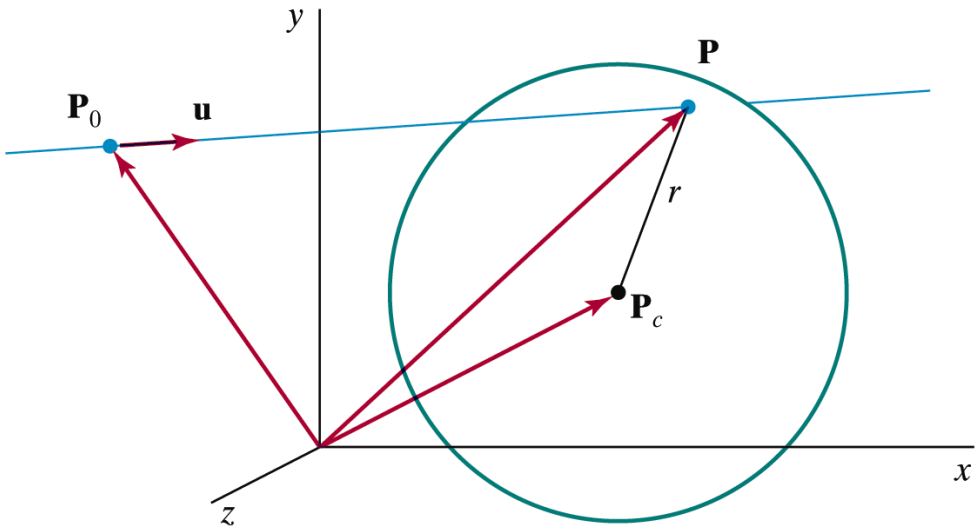
- But at intersection, point must also satisfy ray equation so

$$|P_{\text{pix}} + su - P_c|^2 - r^2$$

- Let $\Delta P = P_c - P_{\text{pix}}$ and, remembering that 1) $|v|^2 = v \cdot v$ for a vector v, and 2) $|u| = 1 = |u|^2$

Ray-Sphere Intersections

- Given a sphere of radius r and centre P_c and a point P on sphere



- The point P must satisfy

$$|P - P_c|^2 - r^2 = 0$$

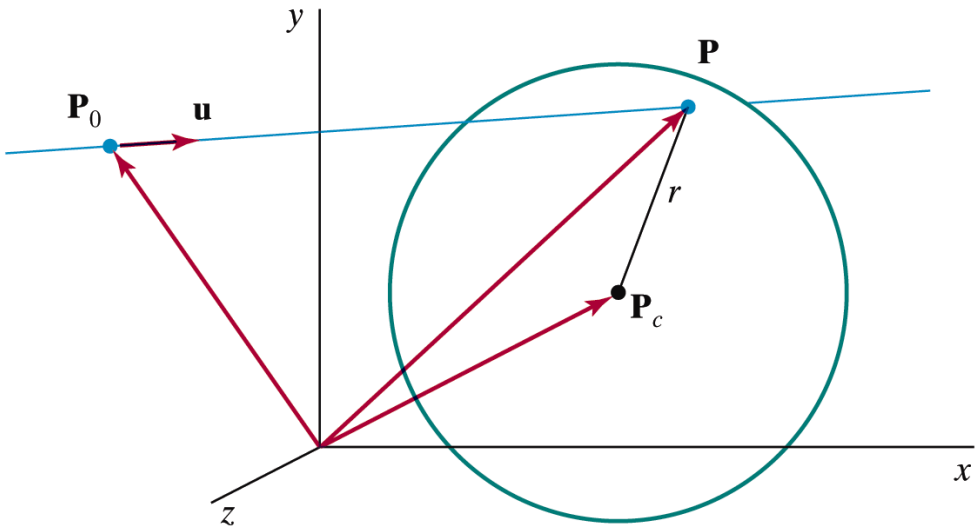
- But at intersection, point must also satisfy ray equation so

$$|P_{\text{pix}} + su - P_c|^2 - r^2$$

- Let $\Delta P = P_c - P_{\text{pix}}$ and, remembering that 1) $|v|^2 = v \cdot v$ for a vector v, and 2) $|u| = 1 = |u|^2$

Ray-Sphere Intersections

- Given a sphere of radius r and centre P_c and a point P on sphere



- The point P must satisfy

$$|P - P_c|^2 - r^2 = 0$$

- But at intersection, point must also satisfy ray equation so

$$|P_{\text{pix}} + su - P_c|^2 - r^2$$

- Let $\Delta P = P_c - P_{\text{pix}}$ and, remembering that 1) $|v|^2 = v \cdot v$ for a vector v, and 2) $|u| = 1 = |u|^2$

Ray-Sphere Intersections

- Given a sphere of radius r and centre P_c and a point P on sphere
- The point P must satisfy

$$|P - P_c|^2 - r^2 = 0$$

- But at intersection, point must also satisfy ray equation so

$$|P_{\text{pix}} + su - P_c|^2 - r^2$$

- Let $\Delta P = P_c - P_{\text{pix}}$ and, remembering that 1) $|v|^2 = v \cdot v$ for a vector v, and 2) $|u| = 1 = |u|^2$

$$s^2 - 2(u \cdot \Delta P)s + (|\Delta P|^2 - r^2) = 0$$

- Using quadratic equation, solution for s is

$$s = u \cdot \Delta P \pm \sqrt{(u \cdot \Delta P)^2 - |\Delta P| + r^2}$$

Ray-Sphere Intersections

- Given a sphere of radius r and centre P_c and a point P on sphere
- The point P must satisfy

$$|P - P_c|^2 - r^2 = 0$$

- But at intersection, point must also satisfy ray equation so

$$|P_{\text{pix}} + su - P_c|^2 - r^2$$

- Let $\Delta P = P_c - P_{\text{pix}}$ and, remembering that 1) $|v|^2 = v \cdot v$ for a vector v, and 2) $|u| = 1 = |u|^2$

$$s^2 - 2(u \cdot \Delta P)s + (|\Delta P|^2 - r^2) = 0$$

- Using quadratic equation, solution for s is

$$s = u \cdot \Delta P \pm \sqrt{(u \cdot \Delta P)^2 - |\Delta P| + r^2}$$

Ray-Sphere Intersect (contd.)

- If discriminant is negative, there cannot be a solution so ray doesn't intersect sphere
- Otherwise we choose the smaller (closer) of two values from solution
- This point is then where we calculate light intensity, reflection, refraction, etc.

Ray-Sphere Intersect (contd.)

- If discriminant is negative, there cannot be a solution so ray doesn't intersect sphere
- Otherwise we choose the smaller (closer) of two values from solution
- This point is then where we calculate light intensity, reflection, refraction, etc.

Ray-Sphere Intersect (contd.)

- If discriminant is negative, there cannot be a solution so ray doesn't intersect sphere
- Otherwise we choose the smaller (closer) of two values from solution
- This point is then where we calculate light intensity, reflection, refraction, etc.

Ray-Sphere Intersect (contd.)

Importance of intersections of ray and sphere

- Each ray must be tested for intersection against *every* surface (possibly millions) Can account for up to 95% of processing of ray-tracing step
- We want to find the intersections as quickly as possible
- Calculating the intersection of the ray with a sphere is one of the important cases

Ray-Sphere Intersect (contd.)

Importance of intersections of ray and sphere

- Each ray must be tested for intersection against *every* surface (possibly millions)
- We want to find the intersections as quickly as possible
- Calculating the intersection of the ray with a sphere is one of the important cases

Ray-Sphere Intersect (contd.)

Importance of intersections of ray and sphere

- Each ray must be tested for intersection against *every* surface (possibly millions)
- We want to find the intersections as quickly as possible
- Calculating the intersection of the ray with a sphere is one of the important cases
  Because it is the easiest to test, we can enclose other objects, or groups of objects, in spheres and we can **eliminate** from further consideration any "containing sphere" (and its contents!) that doesn't intersect the ray

Ray-Sphere Intersect (contd.)

Importance of intersections of ray and sphere

- Each ray must be tested for intersection against *every* surface (possibly millions)
- We want to find the intersections as quickly as possible
- Calculating the intersection of the ray with a sphere is one of the important cases
  An alternative method to speed up calculations is to recursively decompose the world scene in virtual cubes (octants) and store on a linked list those surfaces (polygons) that the cube contains
  It is "easy" to determine what cubes a ray enters
  Call back in CS4085 when we look at **octrees** and **BSP trees**