EDAN30 Photorealistic Computer Graphics

Seminar 1, 2013

Whitted Ray Tracing

(And then some!)

Magnus Andersson, PhD student (magnusa@cs.lth.se)

Today's Agenda

- Structure of Assignments
- Quick prTracer walkthrough
- Assignment 1 specifics
- Some commentary on BRDFs

Structure of Assignments

- Starting point: Ray tracer *skeleton*
- Evolved throughout the course
- Each assignment will build upon the last
  - *No (or little) ”fresh” code releases in-between assignments!*
  - *I.e. You will continue to use your own code in the following assignments*

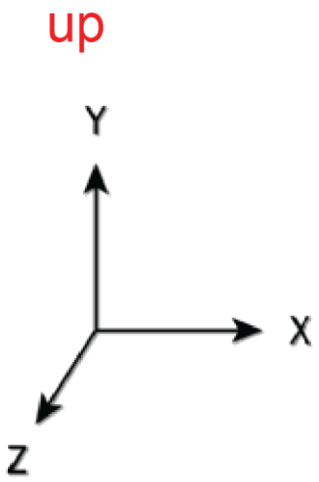
Structure of Assignments

- Starting point: Ray tracer *skeleton*
- Evolved throughout the course
- Each assignment will build upon the last
  - *No (or little) ”fresh” code releases in-between assignments!*
  - *I.e. You will continue to use your own code in the following assignments*

- ***Write maintainable code ;)***

prTracer Overview

- Written in C++
- Mac / Windows
- Right handed coordinate system



- Convenient utility classes
  - *Vector, Matrix, Point, Color, Image, ...*
- Output format
  - *PNG*

Color class

RGB Color

```
Color a = Color(1.0f, 0.0f, 0.0f);
Color b(0.3f, 0.9f, 0.4f);
```

Arithmetic operations

```
Color c, d;
d = a * b;          // Component-wise multiplication
c = (a + b) / 2.0f; // Average of two colors
```

Vector class

Direction in 3D space

```
Vector3D w = (0.5f, 1.0f, 0.3f);
Vector3D v(1.0f, 0.7f, 1.5f);
```

Intuitive math operations (+, -, *, /, +=, etc)

```
Vector3D q = 2.0f * u - v + w;
q = v * w;                          // Dot product (.dot())
q = v % w;                          // Cross product (.cross())
q.normalize();                      // Normalize length to 1
```

Point class

Position in 3D space

```
Point3D p = (0.5f, 1.0f, 0.3f);
Point3D q(1.0f, 0.7f, 1.5f);
```

Similar math operations (+, -, *, /, +=, etc)

```
Vector3D d = Vector(0.0f, 10.0f, -5.0f);
Point3D r = p + d;                    // Point = Point + Vector
Vector3D w = r - p;                   // Vector from p to r
```

Common operations

Debug printout

```
std::cout << p << std::endl;          // p can be Point, Vector or Color
```

Element access

```
float x = p.x;                        // Vector or Point
float r = c.r;                        // Color
```

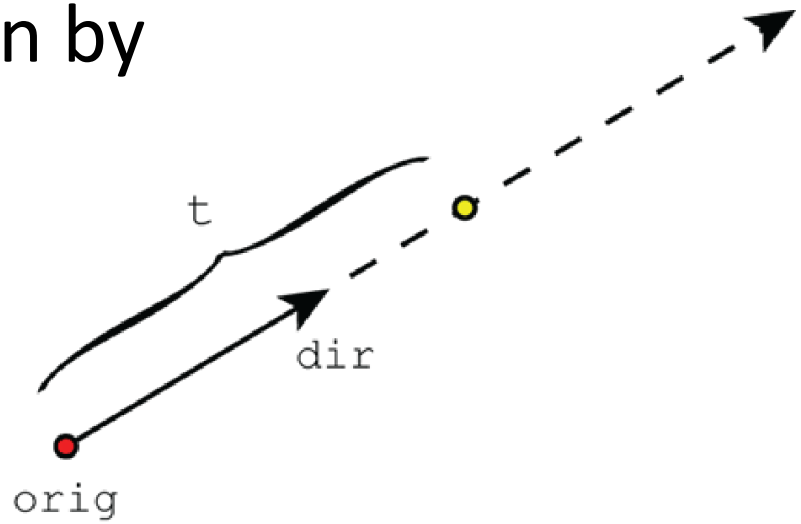
Ray class

Class describing a **ray** in 3D

- Origin (*Point3D*)
- Direction (*Vector3D*)

A point along a ray is given by

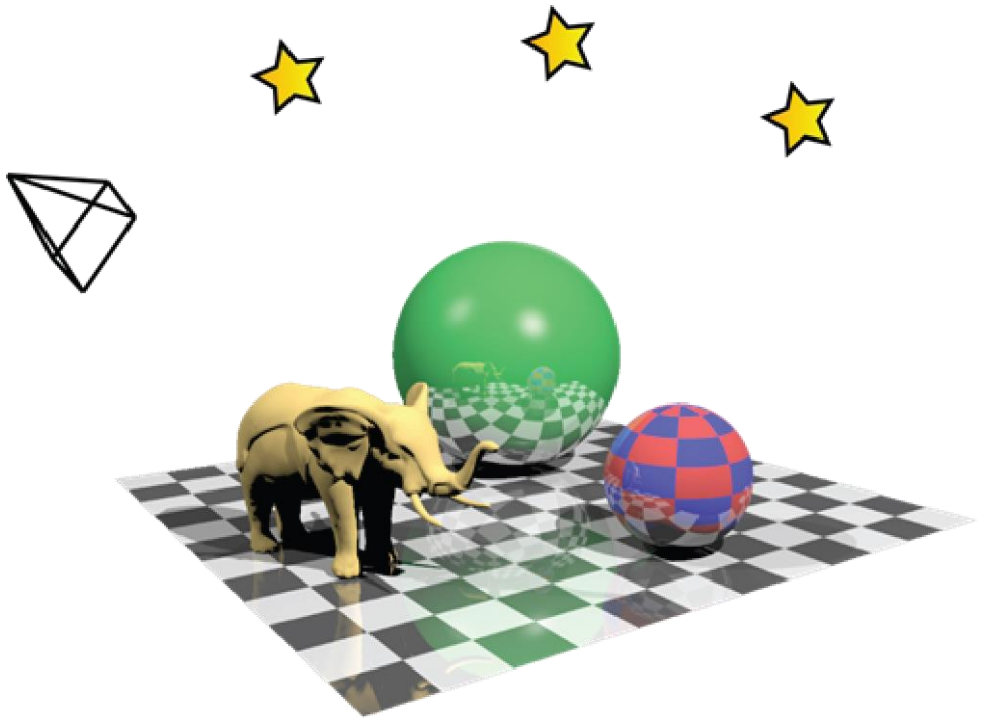
```
Ray r = ...
Point3D p = r.orig + t * r.dir;
```



Scene class

Stores all objects in the 3D world

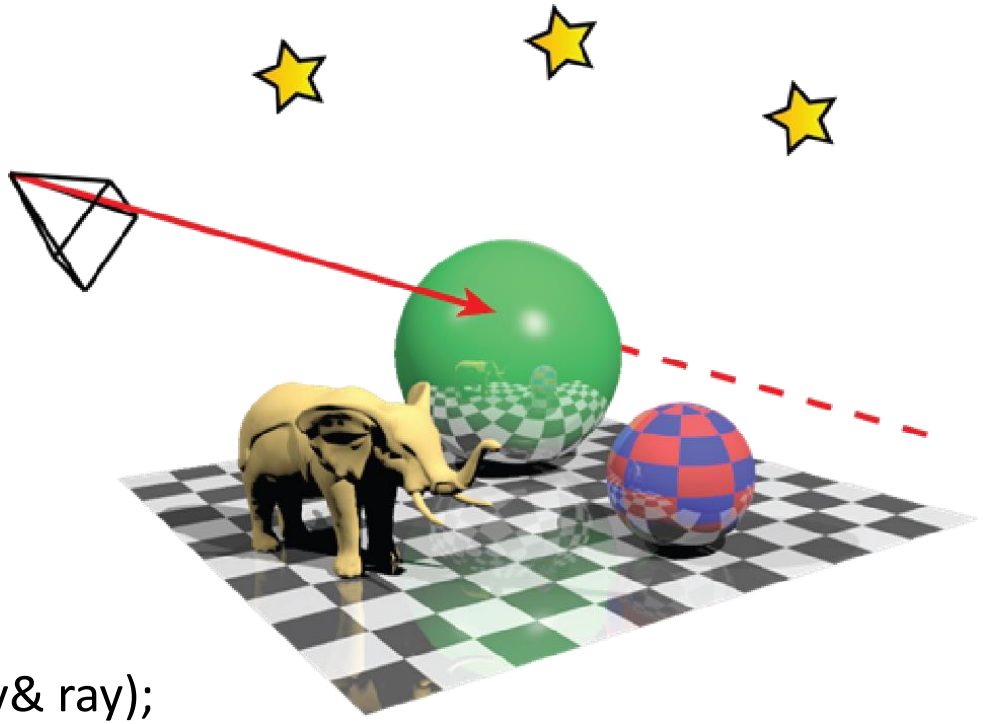
- Primitives
- Light sources
- Cameras



Scene class

Stores all objects in the 3D world

- Primitives
- Light sources
- Cameras



```
bool Scene::intersect(const Ray& ray);
bool Scene::intersect(const Ray& ray, Intersection& is);
```

Ray-Scene Intersection

Boolean test

```
bool Scene::intersect(const Ray& ray);
```

- Is *anything* hit? (Y/N)
- No intersection information
- Used for shadow rays
  - *(we'll get back to this later)*
- Faster

Ray-Scene Intersection

Closest hit test

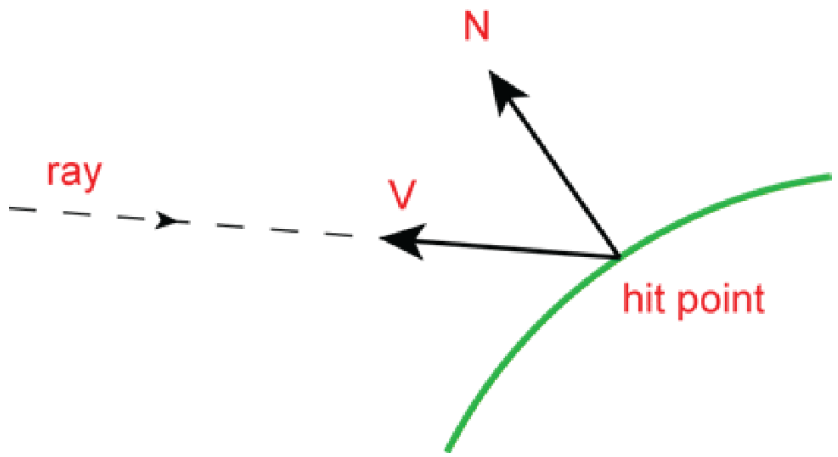
```
bool Scene::intersect(const Ray& ray, Intersection& is);
```

- Returns intersection information
- Used for shading, reflection, refraction, ...
- Slower

Intersection class

Useful information about the intersection

```
class Intersection {
   Point3D      mPosition;      // Position of hit point
   Vector3D     mNormal;        // Surface normal (N)
   Vector3D     mView;          // View direction (V)
   Material     *mMaterial;     // Material of object
   . . .
};
```



Main function

- Build a scene
  - Create materials, objects, lights (scene::add)
- Create image buffer
- Setup camera
- Prepare scene for rendering (Scene::prepare)
- Create ray tracer object
- **Perform ray tracing** (Raytracer::computeImage)
- Save output image

Building a Scene

For this assignment we use the scene

```
buildSpheres(Scene &scene);          // located in main.cpp
```

You are encouraged to play around and building your own scene :)

- Current scene is very programmer artsy

Image and Camera classes

An Image is created by the line

```
Image output(512, 512);          // width x height
```

Camera is setup by

```
Camera cam;
Point3D pos      = Point3D(22.0f, 24.0f, 26.0f);
Point3D target   = Point3D(0.0f, 2.0f, 0.0f);
Vector3D up      = Vector3D(0.0f, 1.0f, 0.0f);
float fov        = 52.0f;
cam->setLookAt(pos, target, up, fov);
scene.add(cam);
```

Rendering a Scene

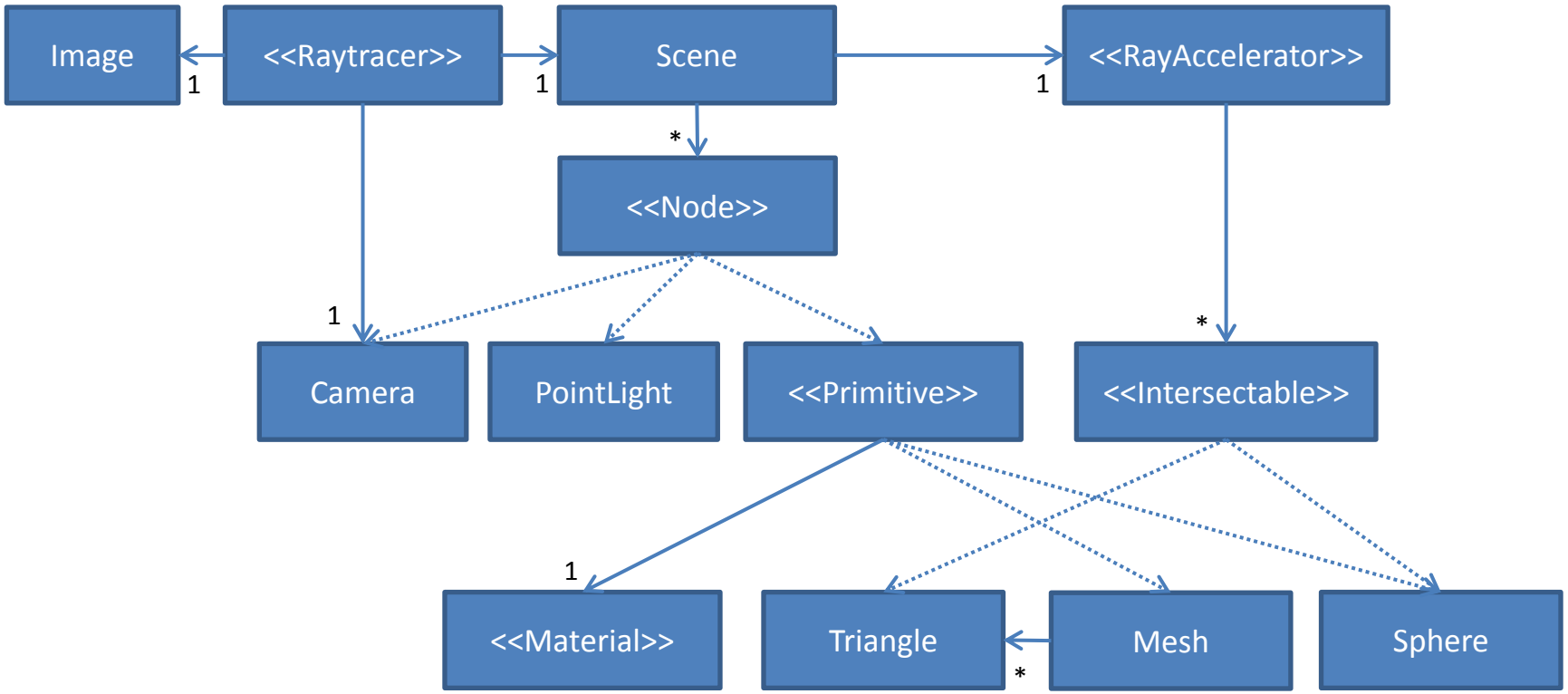
First we must call Scene::prepare()

- Transform to world space
- Set up <<RayAccelerator>>
  - In this lab this is just a list of intersectables

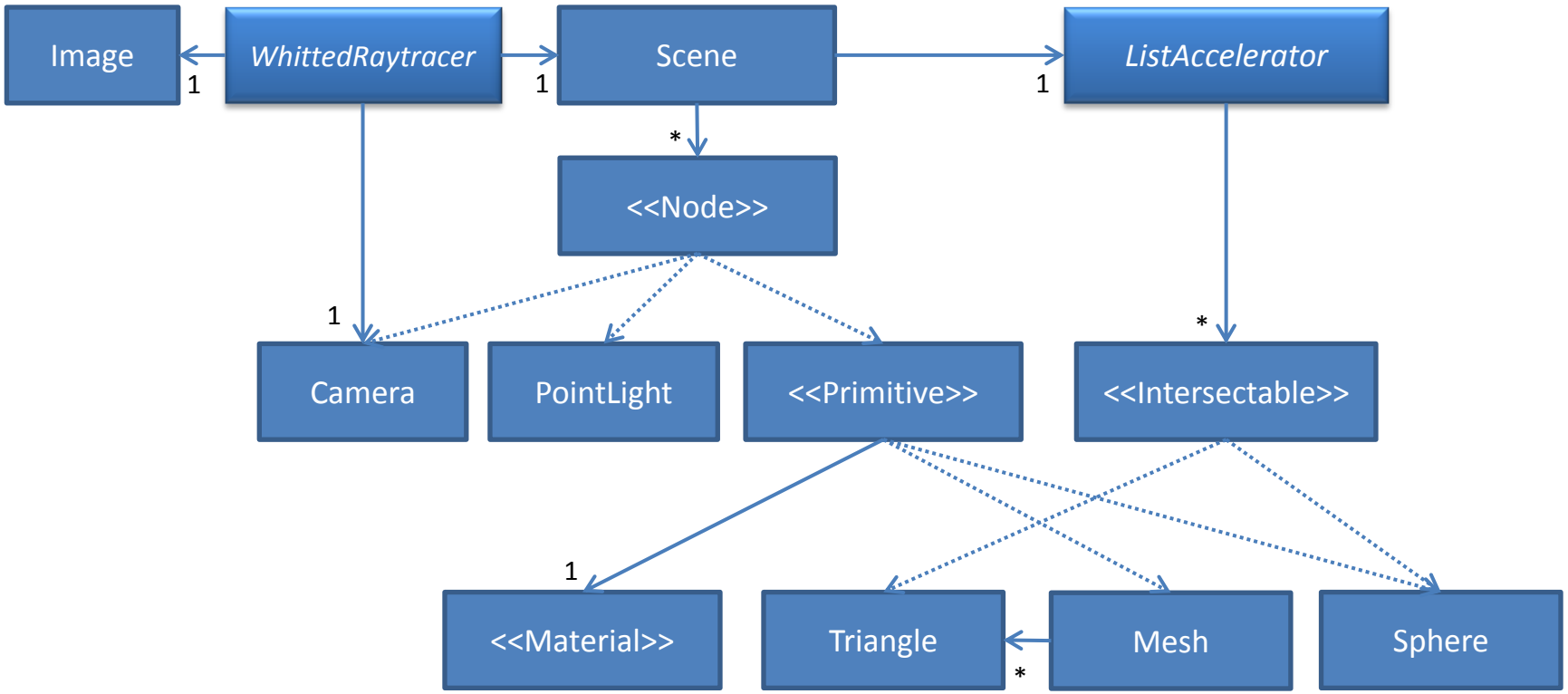
Create <<Raytracer>> object and start rendering

```
WhittedTracer rt(&scene, &output);
rt.computeImage();
output.save("output.png");
```

prTracer Skeleton Overview



prTracer Skeleton Overview

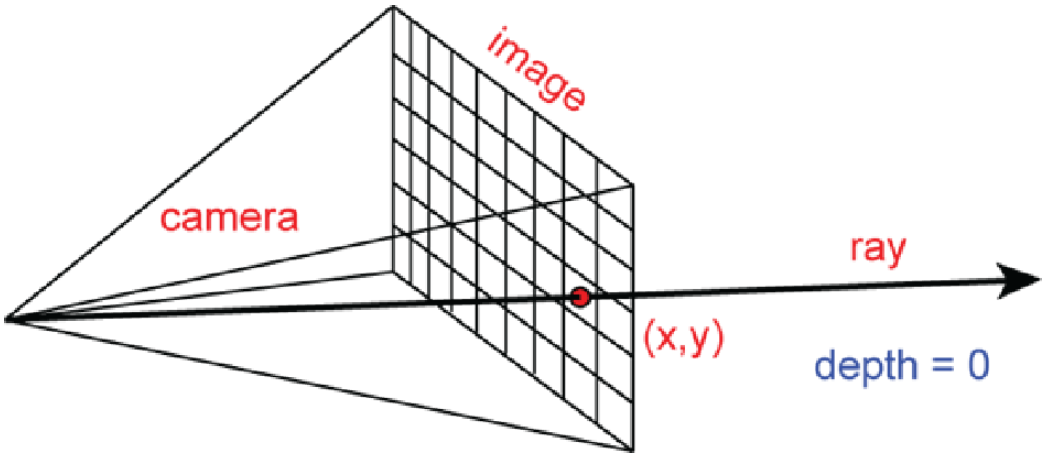


Shooting eye rays

In *WhittedTracer.cpp*

computeImage() calls **tracePixel()** for each pixel (x, y).

tracePixel() calls **trace()** for rays in the pixel.



Assignment 1

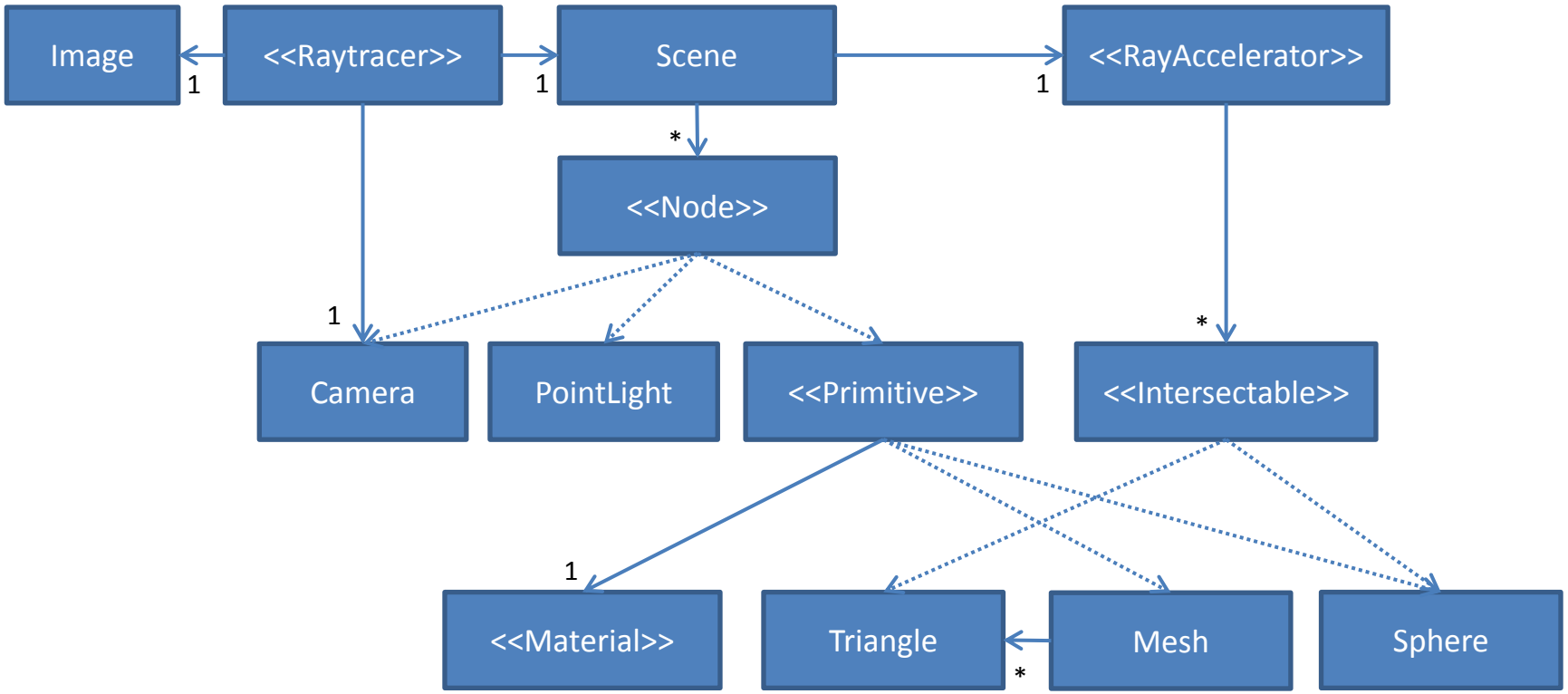
- Diffuse Reflection
- Whitted ray tracing
  - Shadows
  - Reflections
  - Refractions
- Super sampling
- Blinn-Phong Shading *(Optional)*
- Ray-Triangle Intersection

Default Implementation

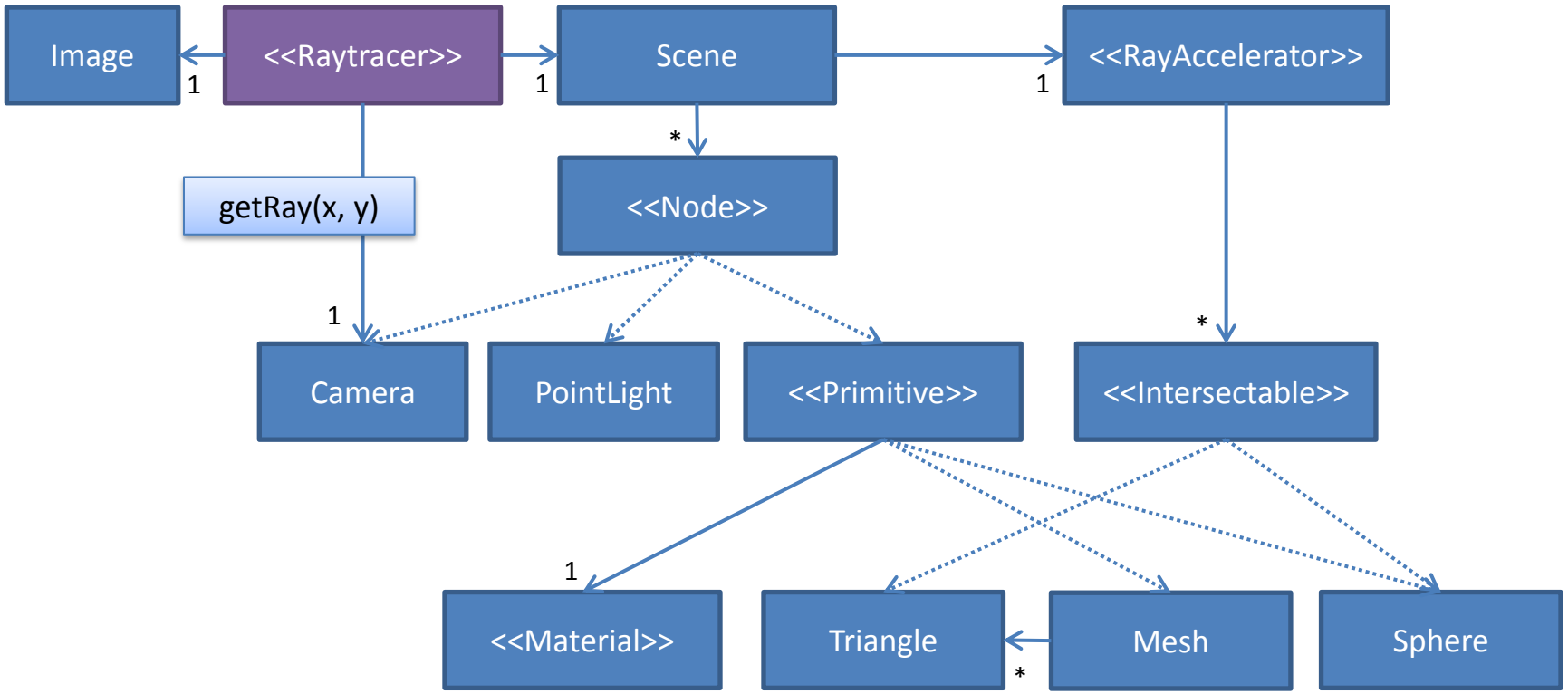
prTracer embryo

- Colors the pixel white if the ray hits anything, black otherwise
- This is what you should get when you compile and run your code for the first time

Program flow

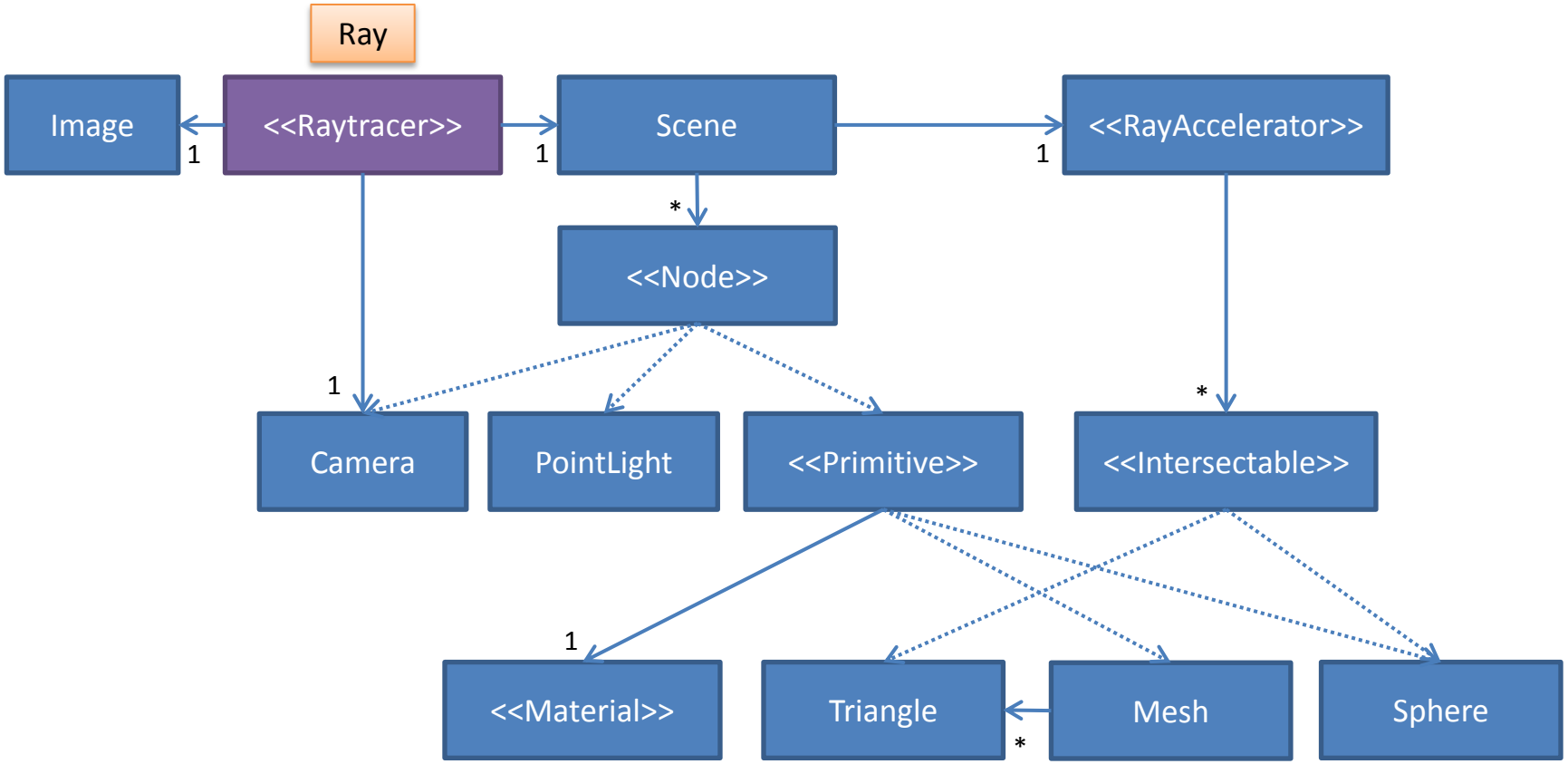


Program flow

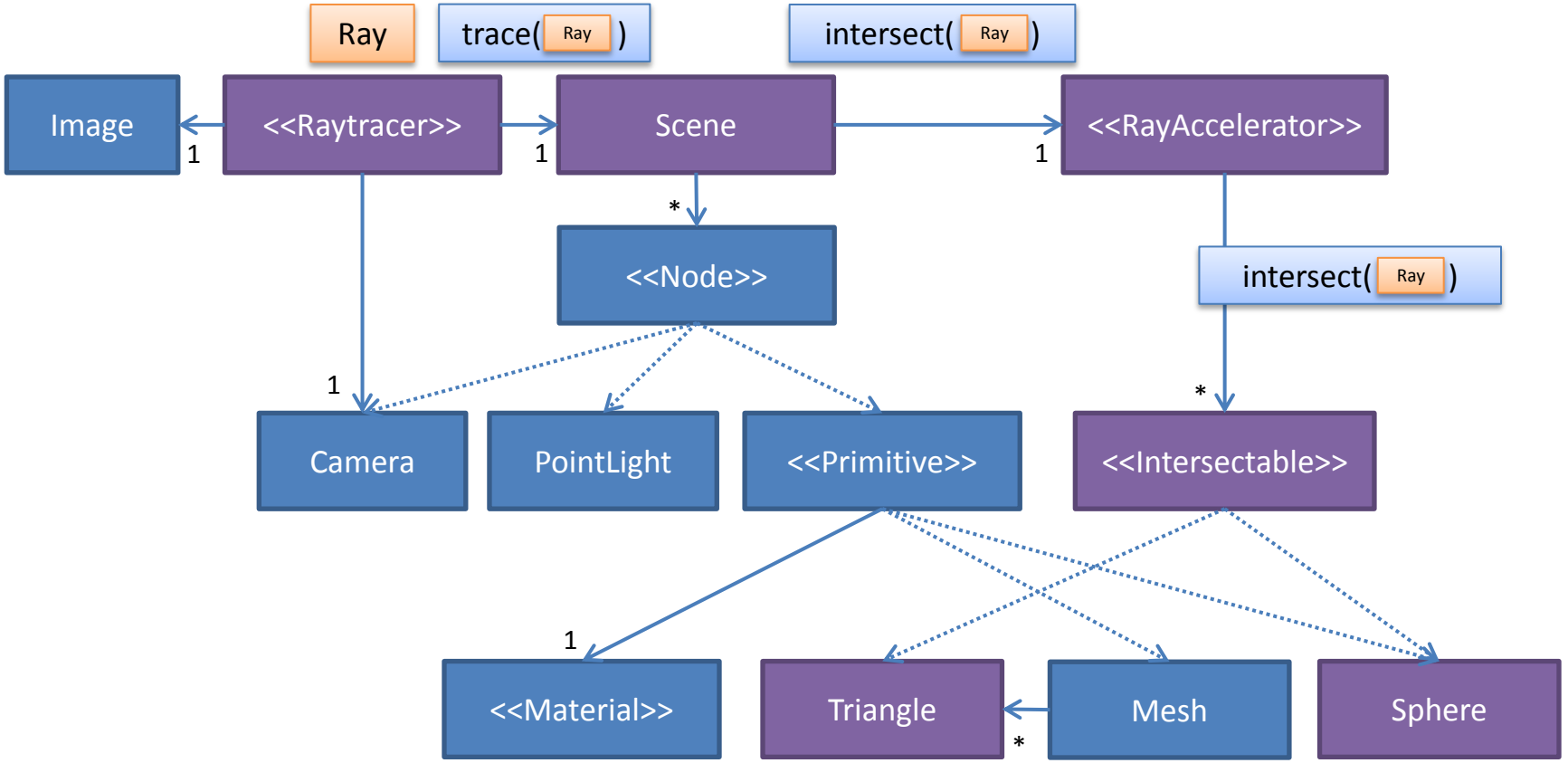


Fetch view ray from camera

Program flow

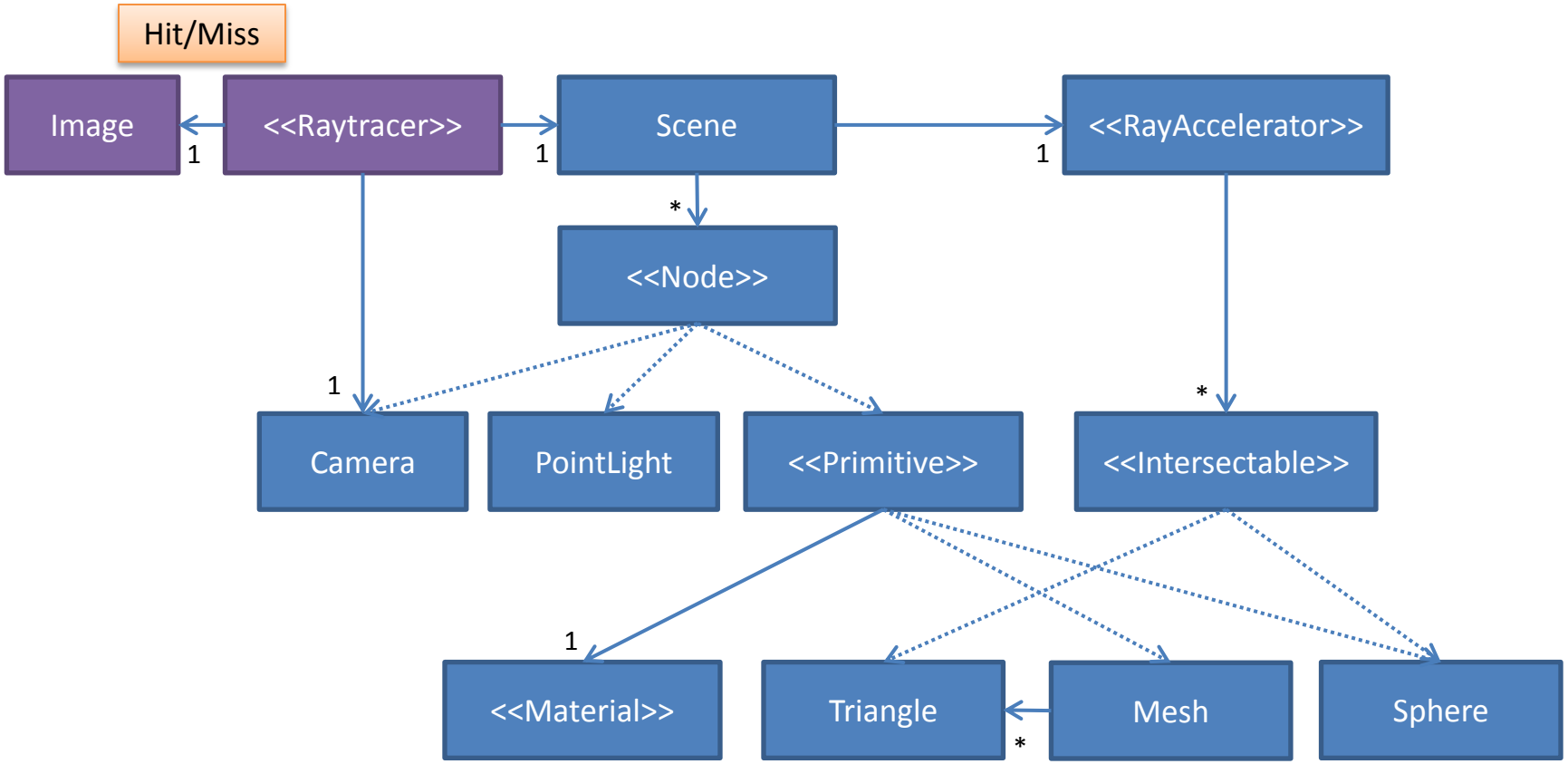


Program flow



Intersect with the scene's intersectables

Program flow



Hit = write white pixel Miss = write black pixel

Assignment 1

- Diffuse Reflection
- Whitted ray tracing
  - Shadows
  - Reflections
  - Refractions
- Super sampling
- Ray-Triangle Intersection
- Blinn-Phong Shading *(Optional)*

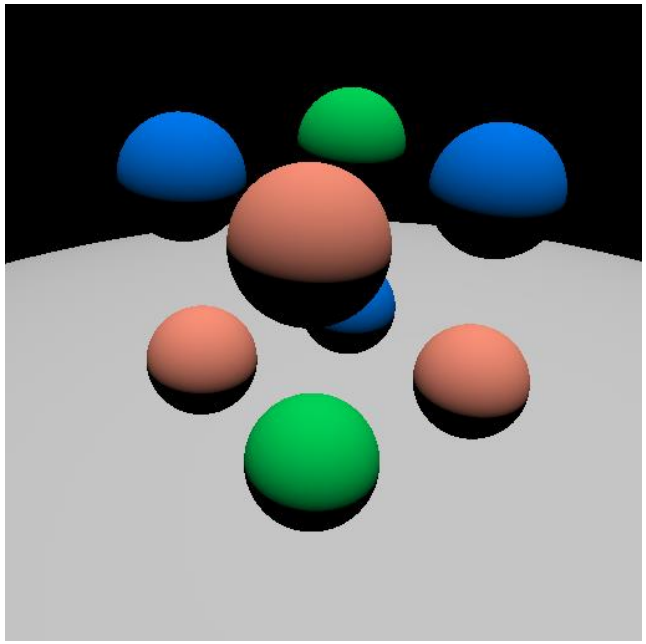
Assignment 1

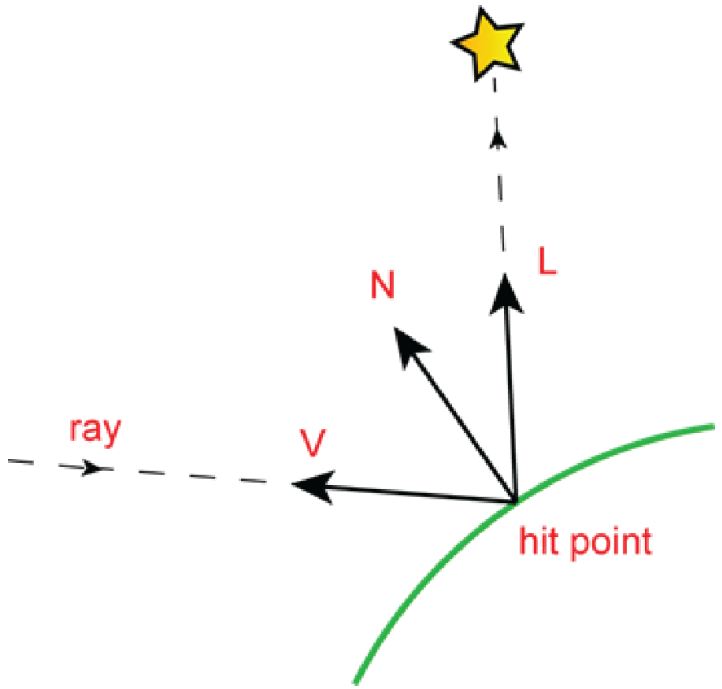
- Diffuse Reflection
- Whitted ray tracing
  - Shadows
  - Reflections
  - Refractions
- Super sampling
- Ray-Triangle Intersection
- Blinn-Phong Shading *(Optional)*

Diffuse Reflection

For this assignment you'll need to modify

Color WhittedTracer::trace(const Ray& ray, int depth)





Diffuse Reflection

$$L_{out}(x \rightarrow \Theta) = L_{in}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi)$$

- Incoming radiance — light->getRadiance();
- BRDF — is.mMaterial.evalBRDF(is, Ψ);
- Incident angle — max(Ψ.dot(is.mNormal), 0.0f);

Diffuse Reflection

The scene has a number of light sources

```
int n = mScene->getNumberOfLights();
PointLight *light = mScene->getLight(i);
```

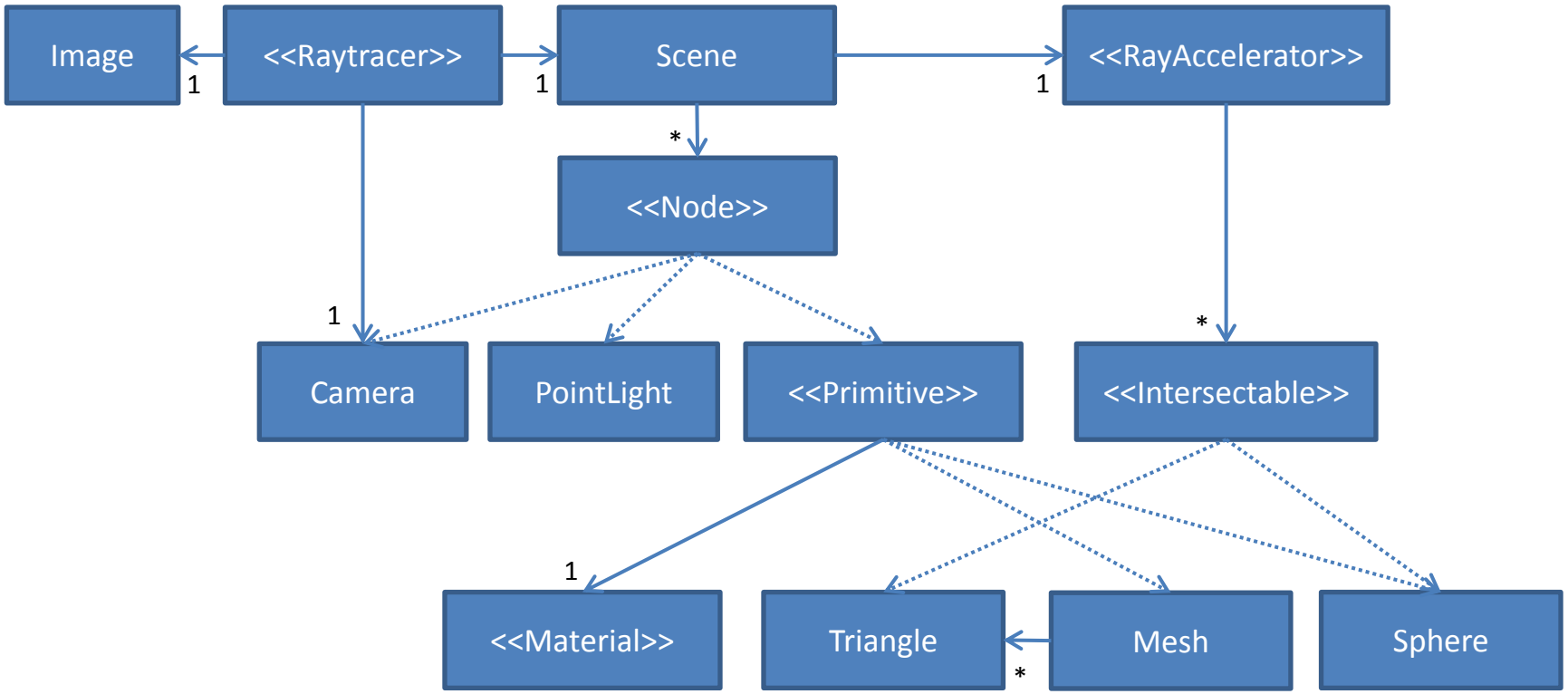
Calculating the irradiance (light) vector (Ψ)

```
Vector3D lightVec = light->getWorldPosition() – is.mPosition;
lightVec.normalize();
```

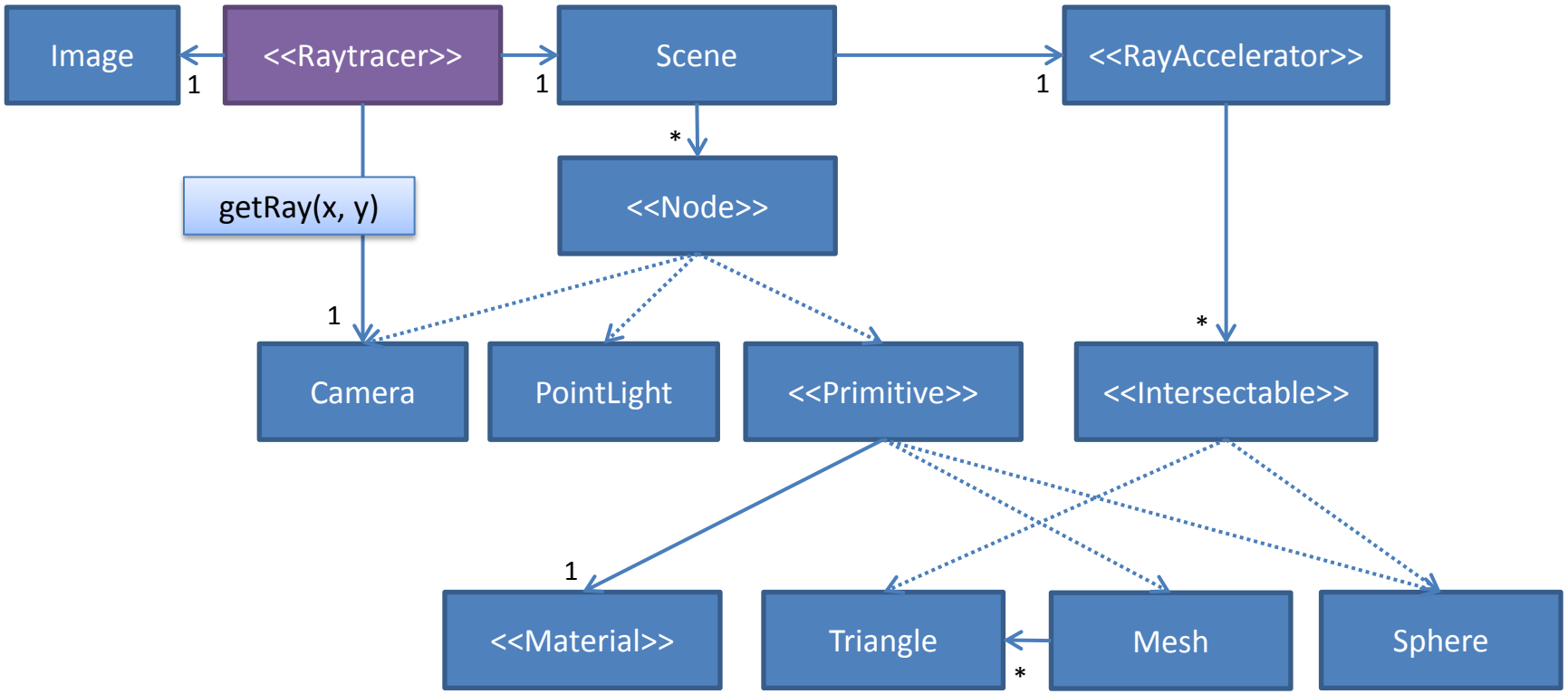
Sum the contribution to get the right result

$$L_{out}(x \rightarrow \Theta) = \sum_{i}^{lights} L_i(x \leftarrow \Psi_i) f_r(x, \Psi_i \leftrightarrow \Theta) cos(N_x, \Psi_i)$$

Program flow

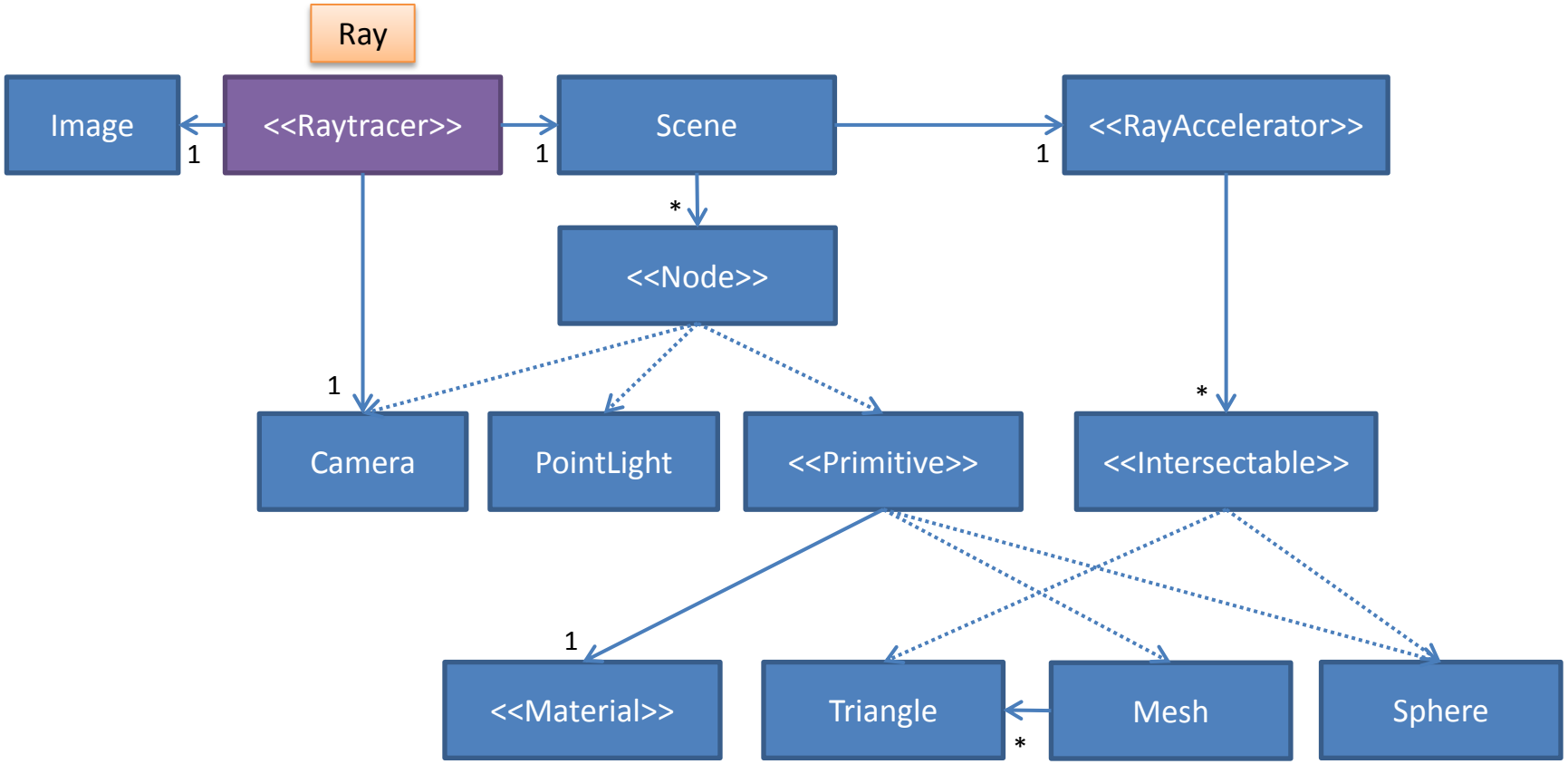


Program flow

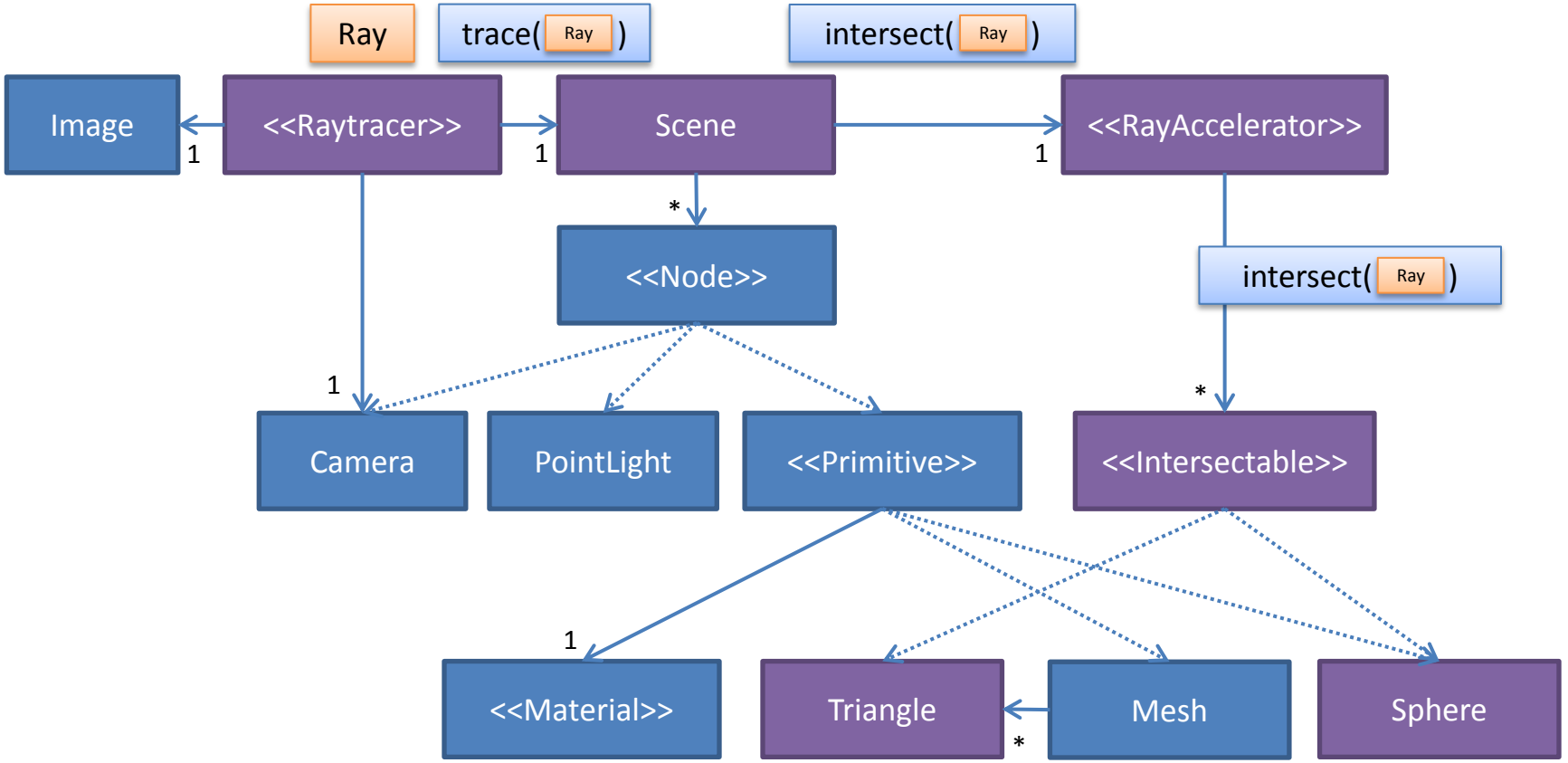


Fetch view ray from camera

Program flow

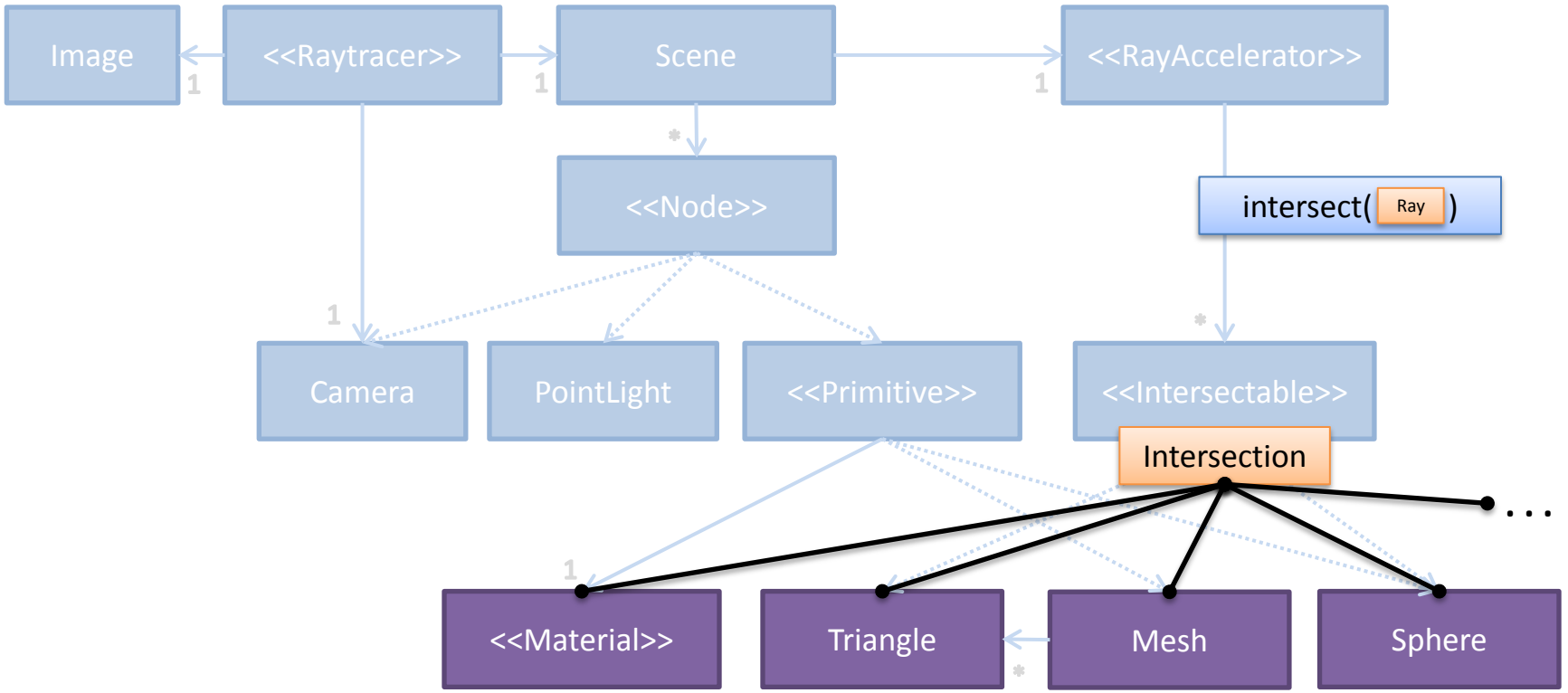


Program flow



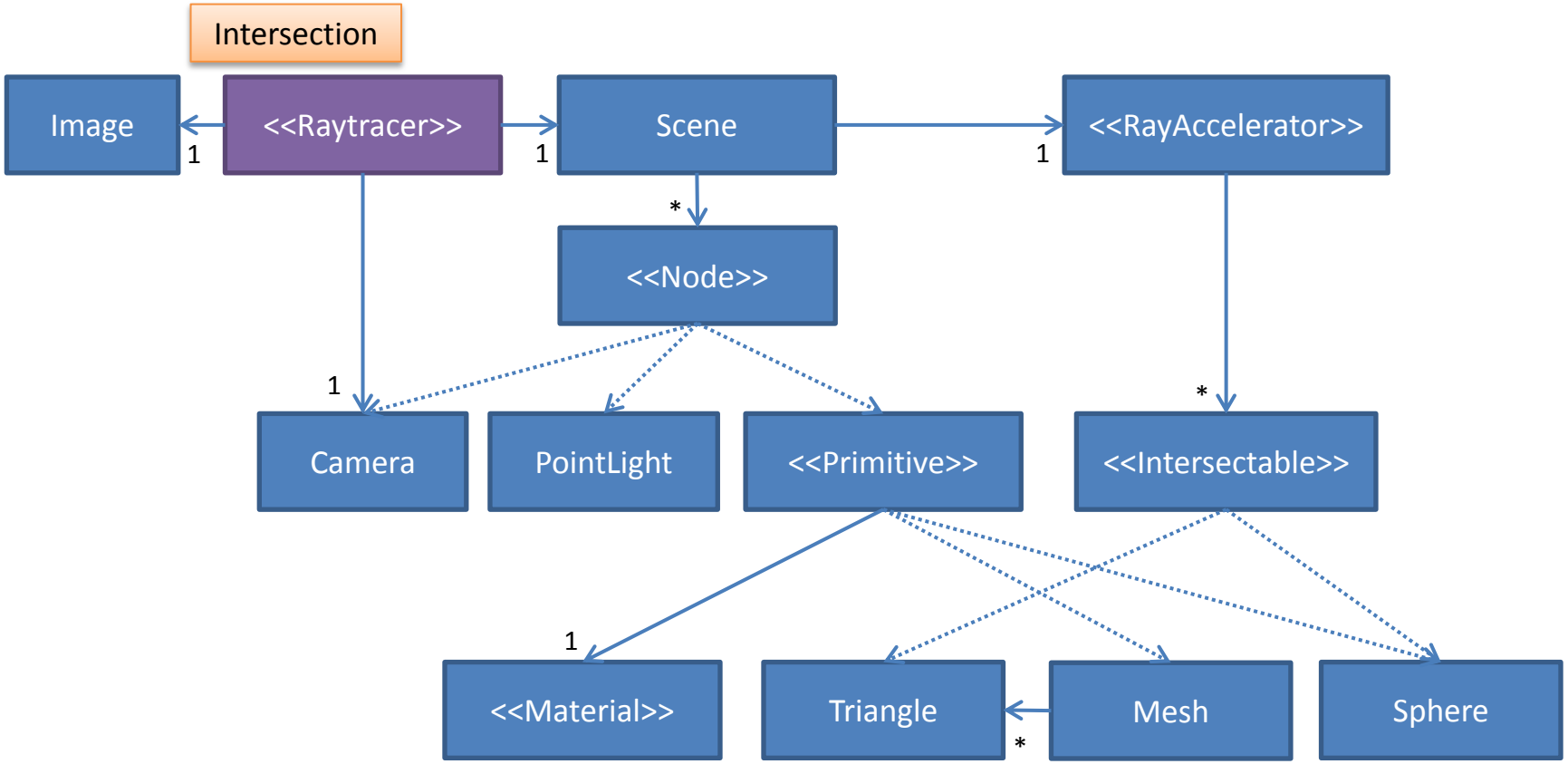
Intersect with the scene's intersectables

Program flow



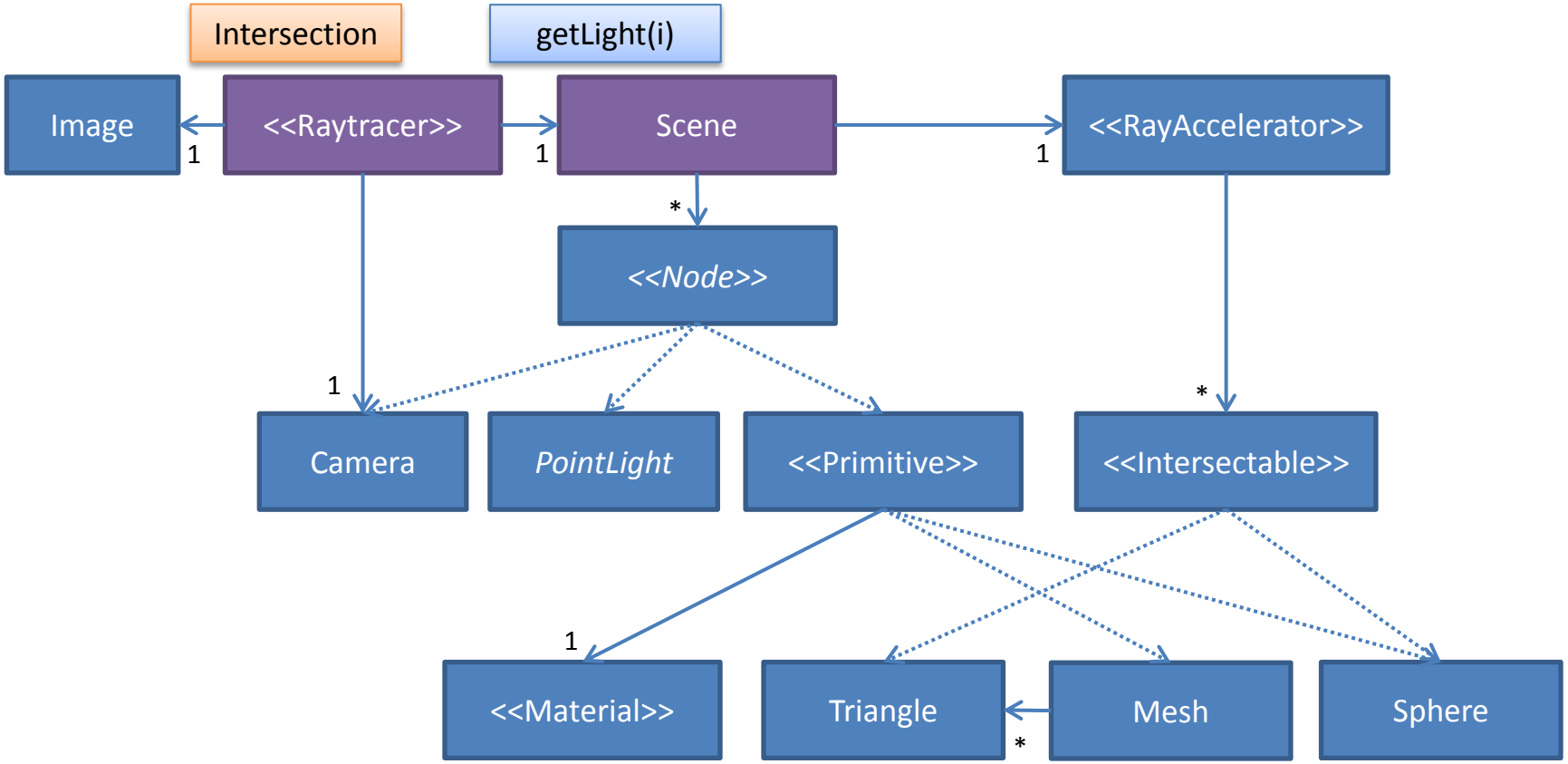
Gather hit point information

Program flow



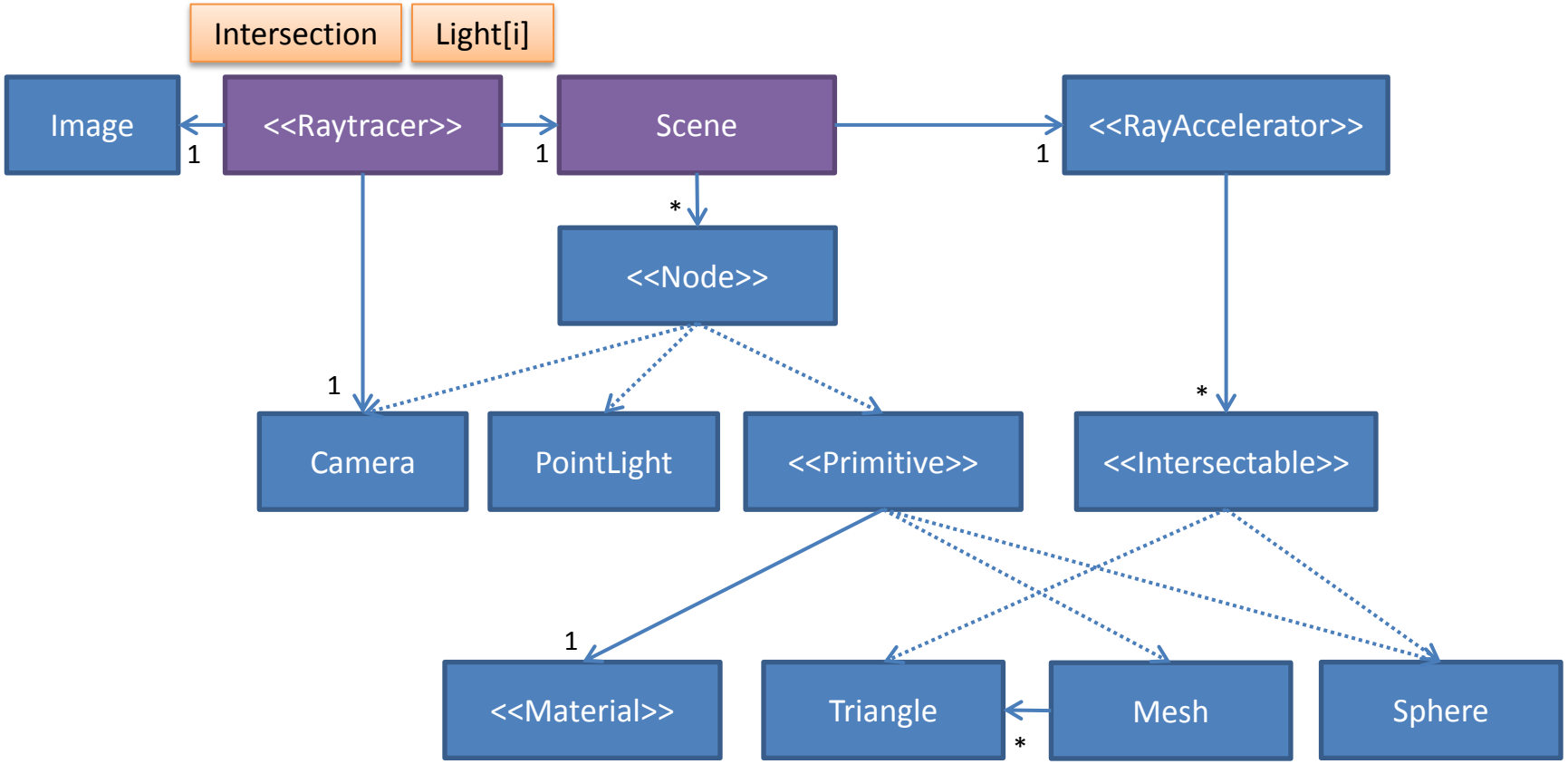
Intersect with the scene's intersectables

Program flow

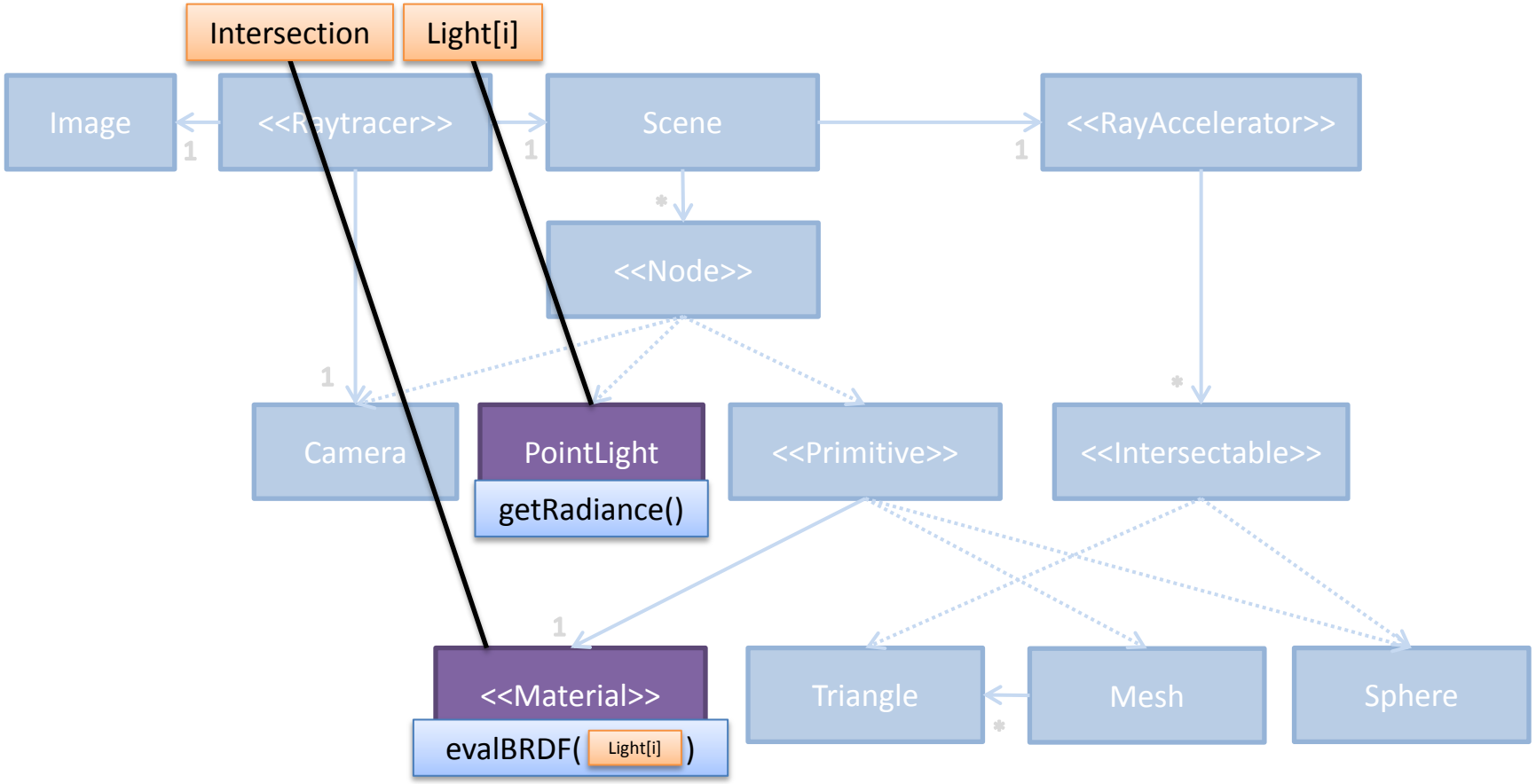


Get light *i* from the scene

Program flow

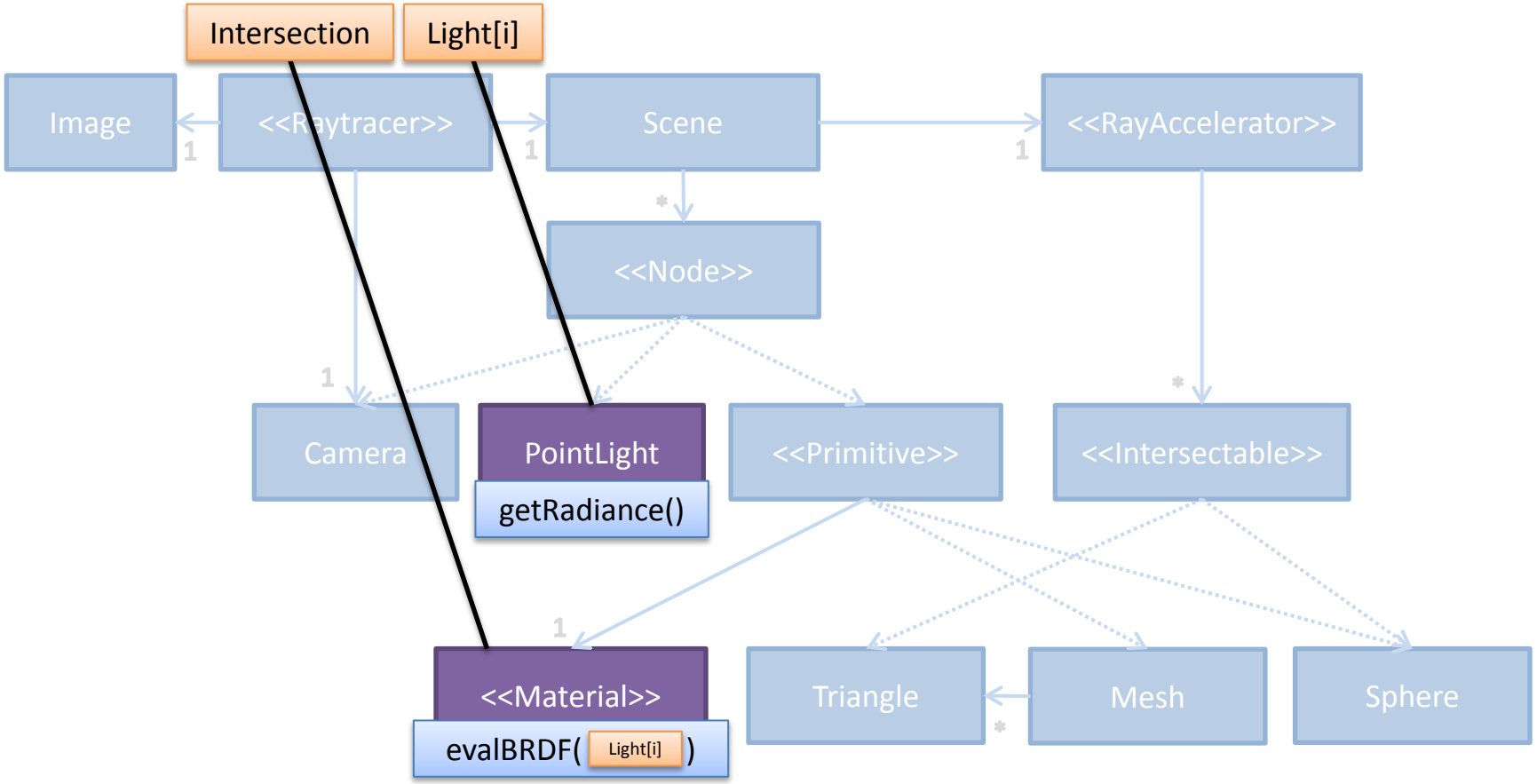


Program flow



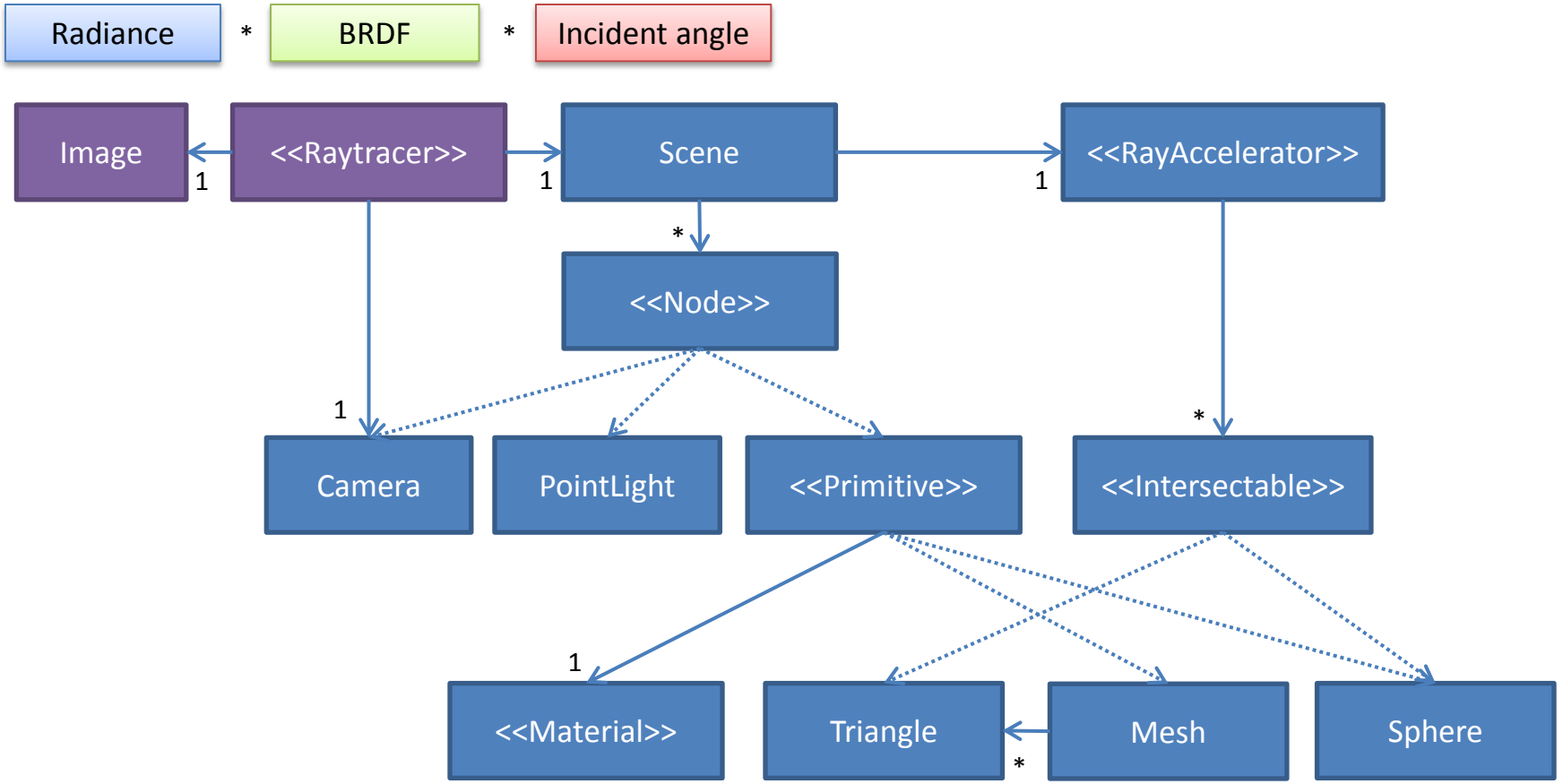
For each light, get light radiance and evaluate material BRDF

Program flow



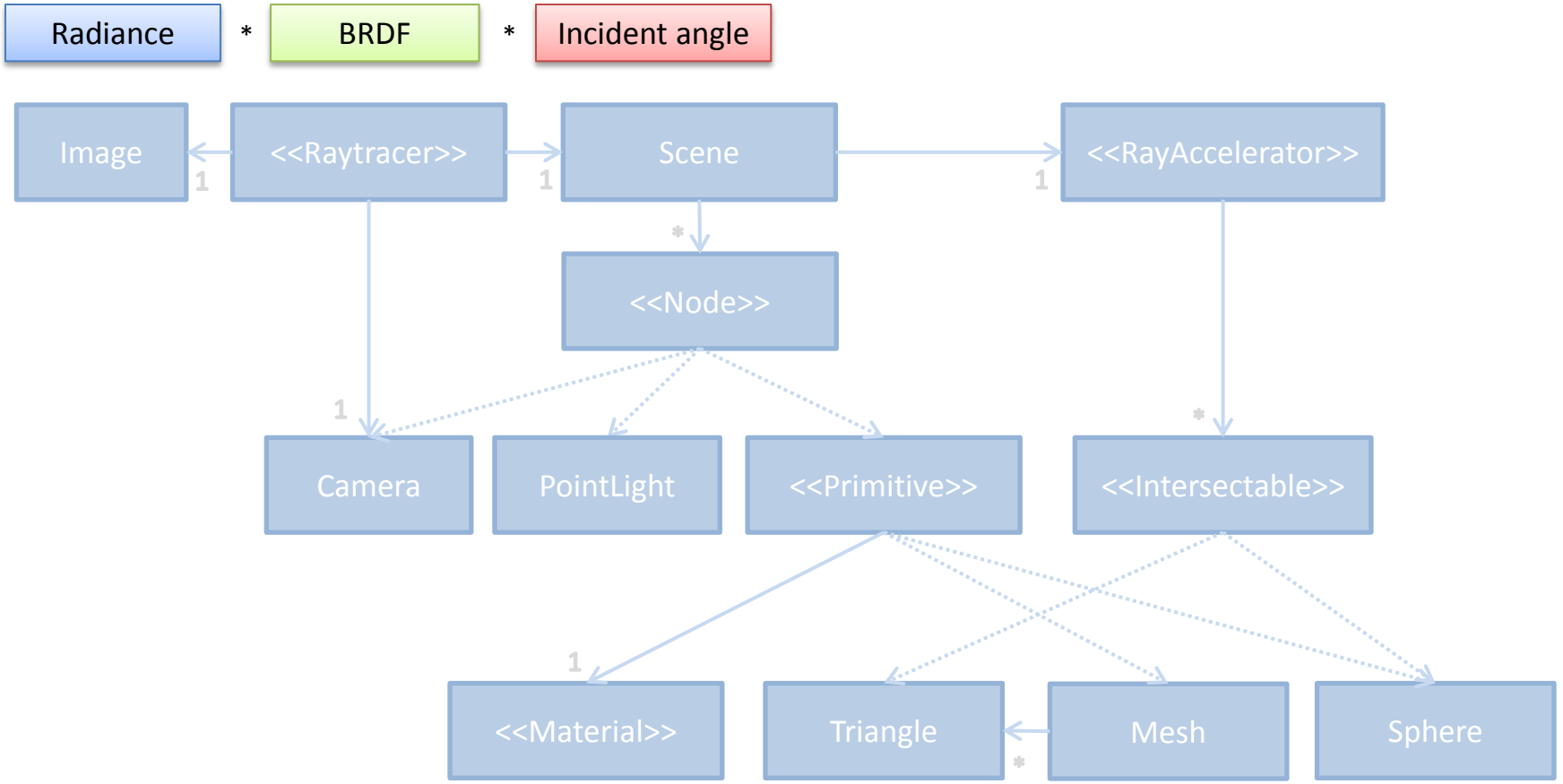
In our case the Material is Diffuse

Program flow



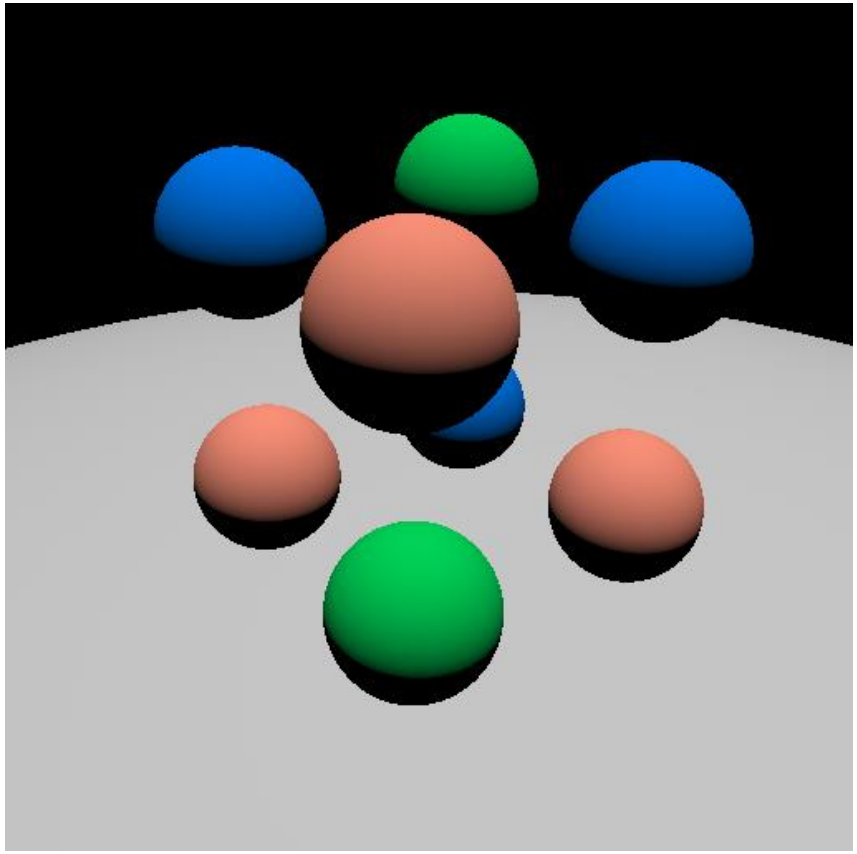
For each light, contribute to the final pixel color

Program flow



$$L_{out}(x \rightarrow \Theta) = L_{in}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi)$$

Diffuse Reflection

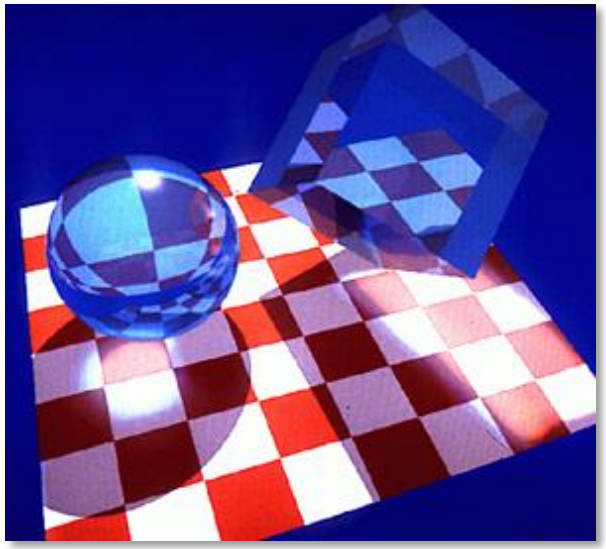


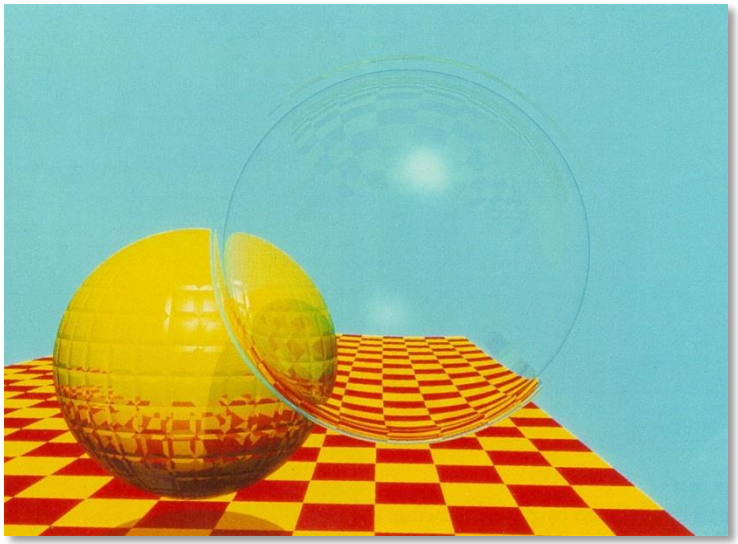
Assignment 1

- Diffuse Reflection
- Whitted ray tracing
  - Shadows
  - Reflections
  - Refractions
- Super sampling
- Ray-Triangle Intersection
- Blinn-Phong Shading *(Optional)*

Whitted Ray Tracing

- Introduced by Turner Whitted in 1980
- Added ***shadows*** and recursive ***reflection*** and ***refraction***
- **Not** physically correct!

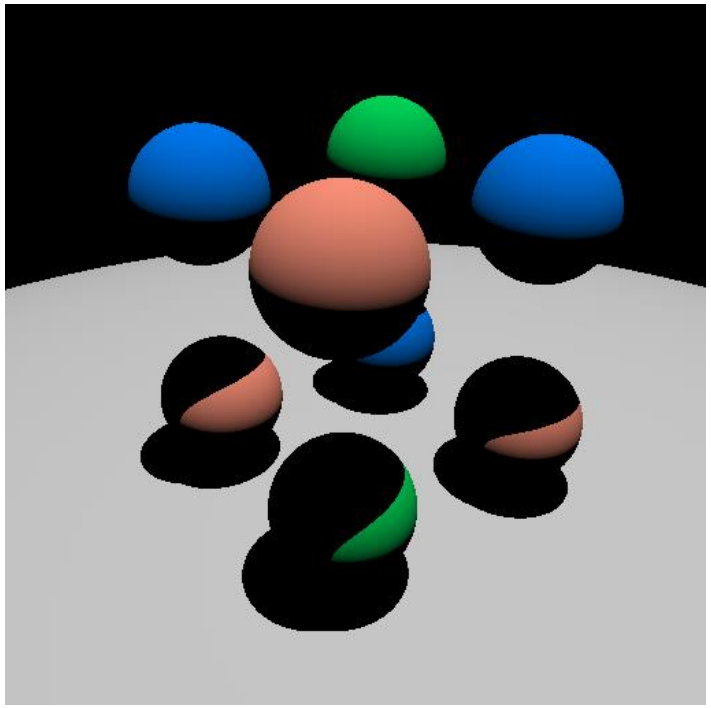


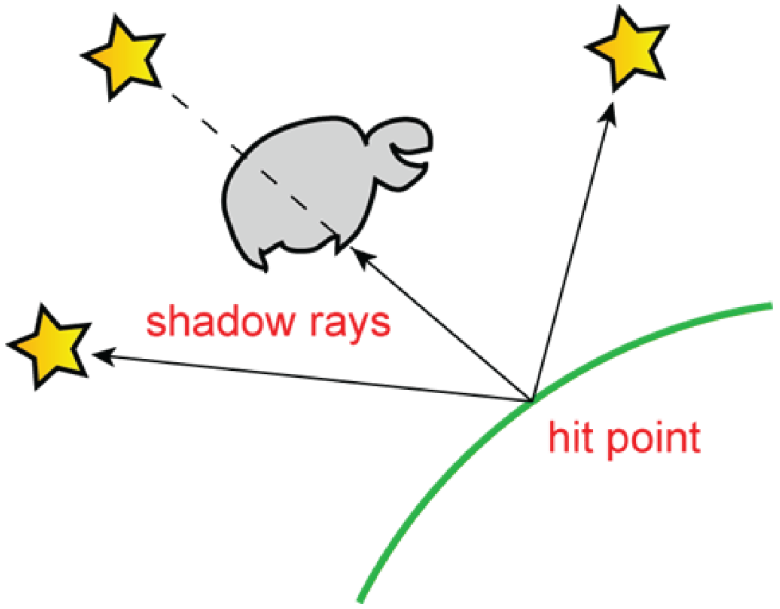


Shadows

For this assignment you will need to modify

Color WhittedTracer::trace(const Ray& ray, int depth)





Shadows

Helper function:

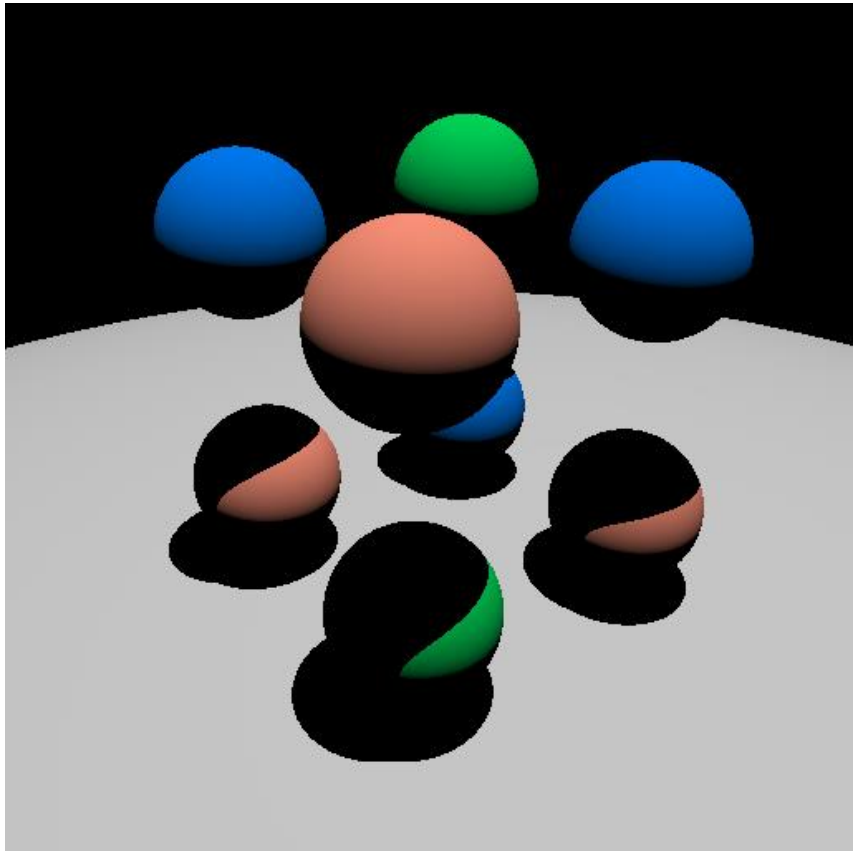
```
PointLight *light = mScene->getLight(i);
Ray r = is.getShadowRay(light);
```

Returns a shadow ray:

- r.orig = is.mPosition + *epsilon* * is.mNormal
- r.dir = direction of light vector
- $r.t_{max}$ = distance from hit point to light source

(you don't need to implement this!)

Shadows

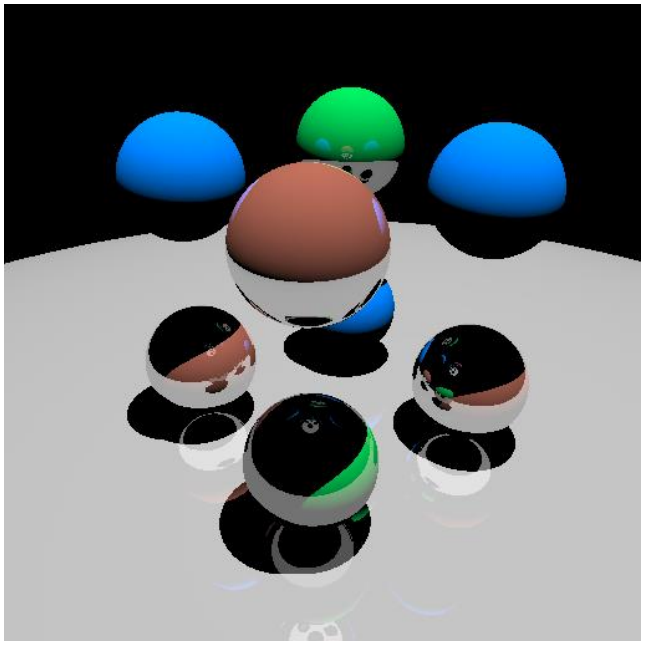


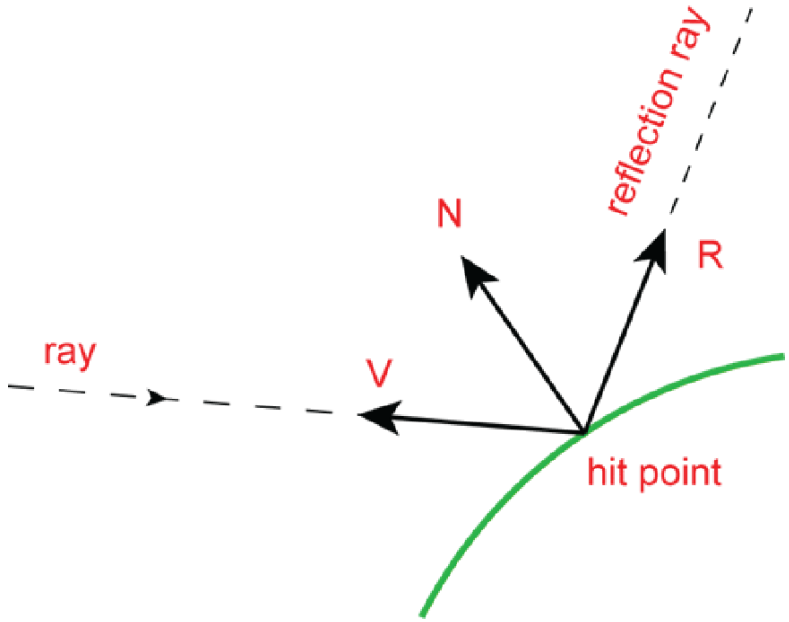
Reflection

For this assignment you will need to modify

Ray Intersection::getReflectedRay() const

Color WhittedTracer::trace(const Ray& ray, int depth)

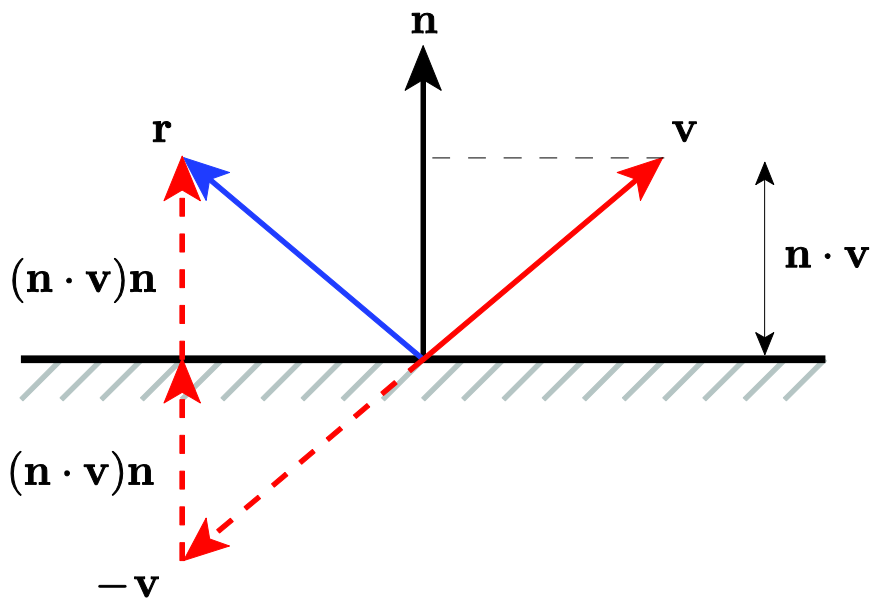




Reflection

First, add proper reflection code to

Ray Intersection::getReflectedRay() const

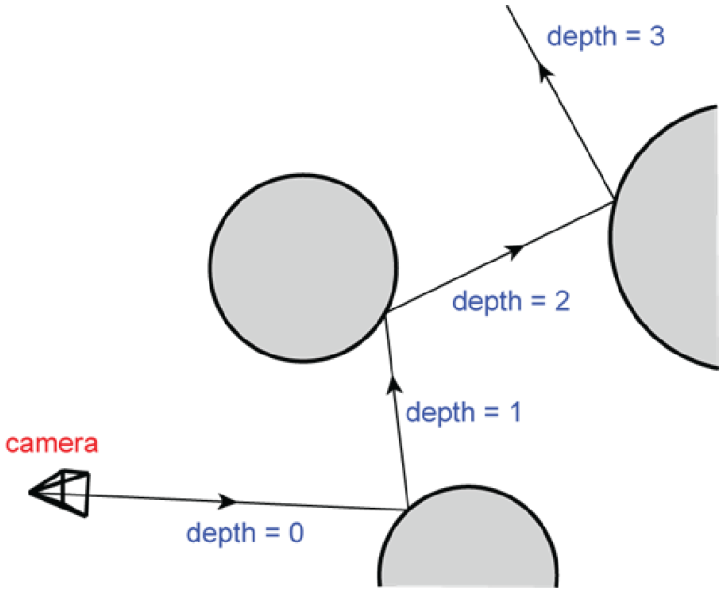


$$\mathbf{r} = 2(\mathbf{n}\cdot\mathbf{v})\mathbf{n} - \mathbf{v}$$

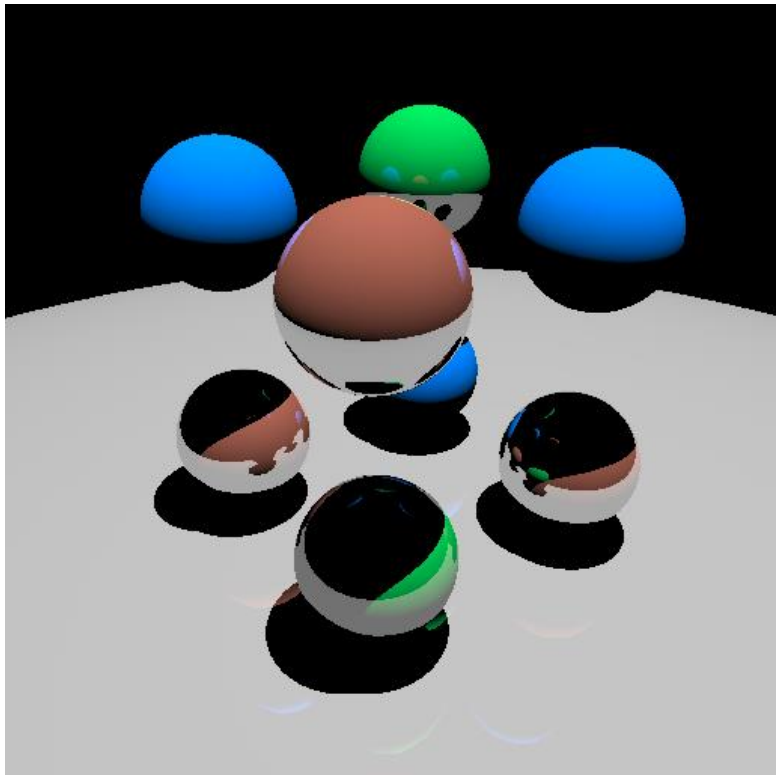
Reflection

Next, modify Color WhittedTracer::trace(const Ray& ray, int depth) to follow reflection rays *recursively*.

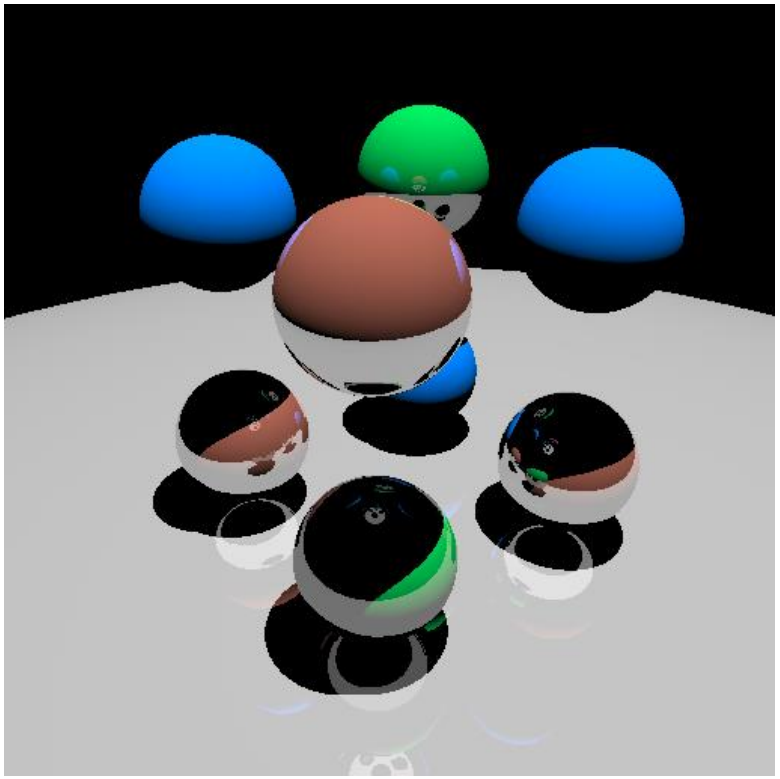
– Use fixed recursion depth (2 or 3 to start with)



Reflection



1 Bounce



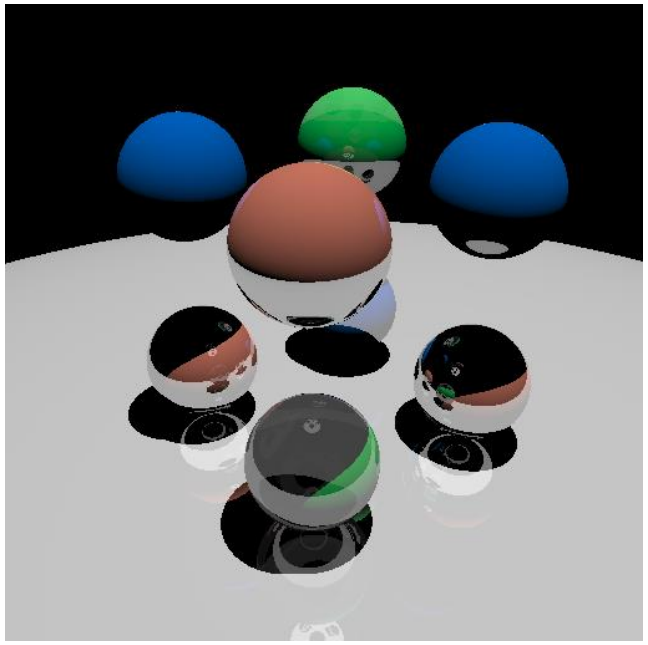
2 Bounces

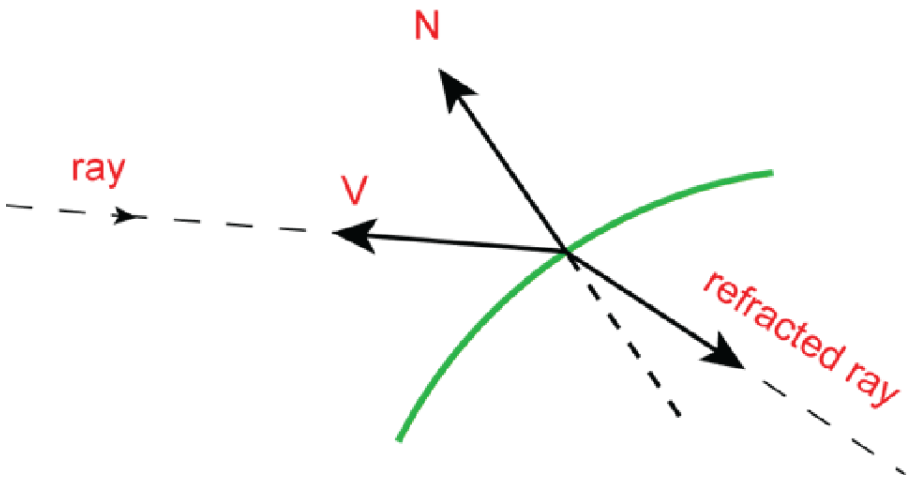
Refraction

For this assignment you will need to modify

Ray Intersection::getRefractedRay() const

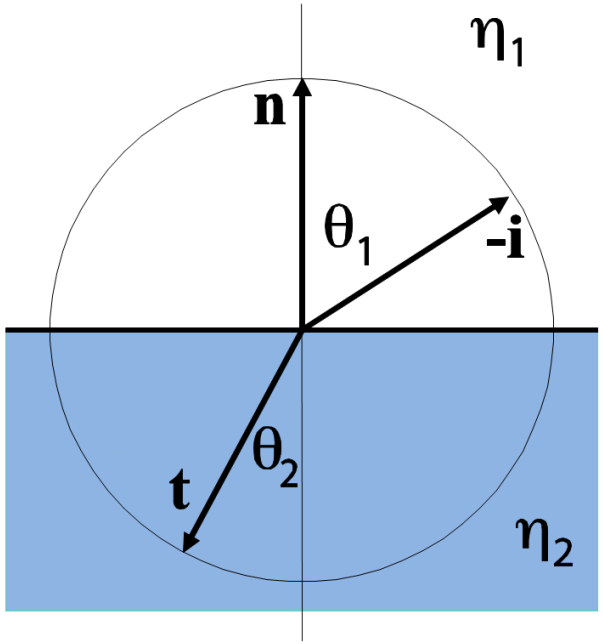
Color WhittedTracer::trace(const Ray& ray, int depth)





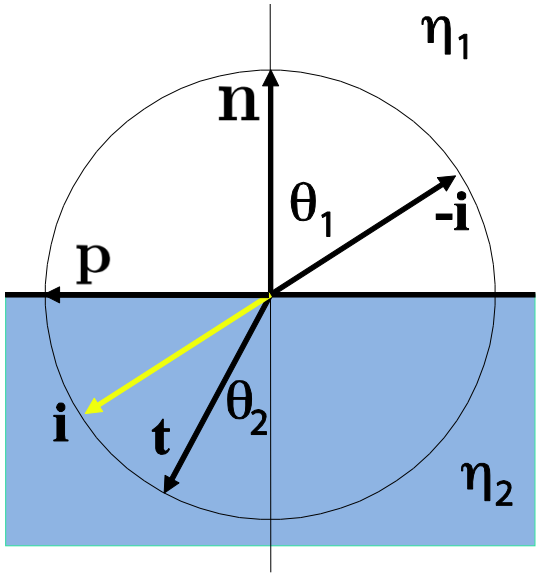
Refraction

Modify Ray Intersection::getRefractedRay() const to properly calculate the refraction vector **t**

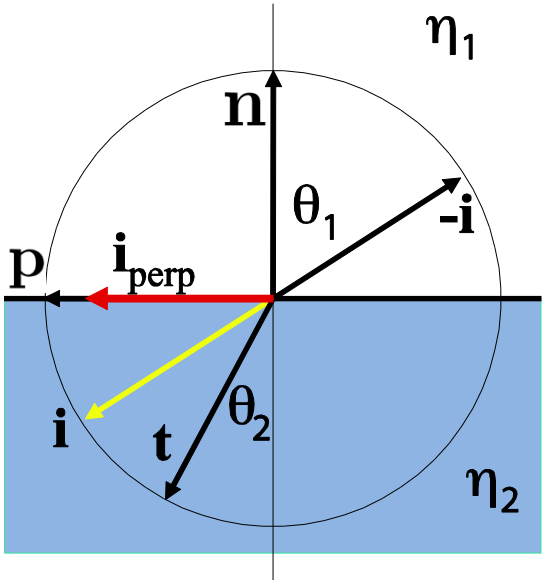


→→→ (Note: -**i** = **Ψ**) ←←←

Refraction



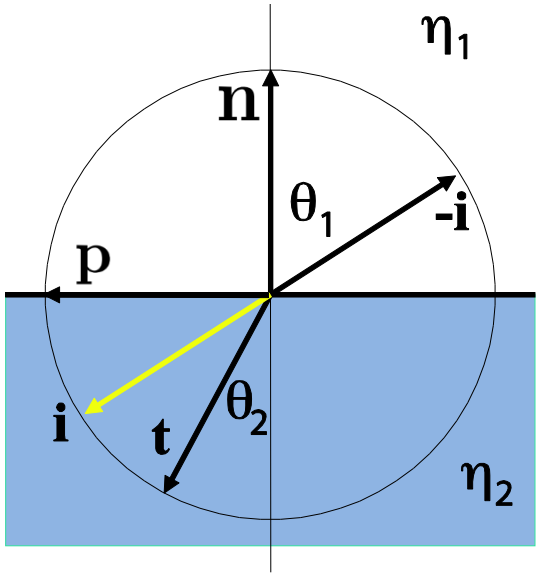
Find vectors in the refraction plane to express **t** with:

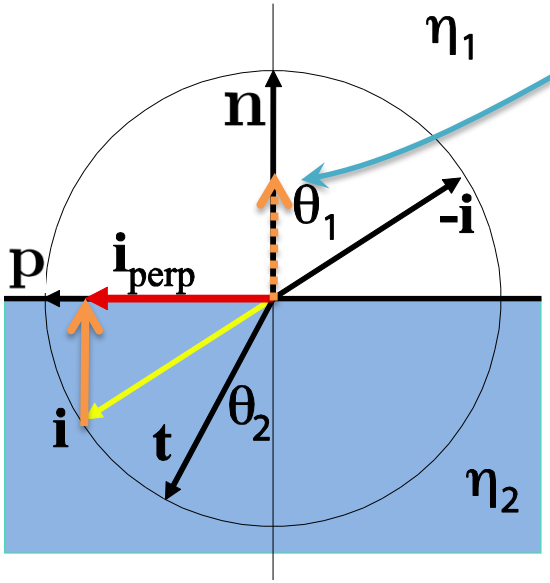


$$\mathbf{i_{perp}} = \mathbf{i} + \cos\Theta_1 \mathbf{n}$$

$$\mathbf{p} = \frac{\mathbf{i_{perp}}}{||\mathbf{i_{perp}}||}$$

Refraction



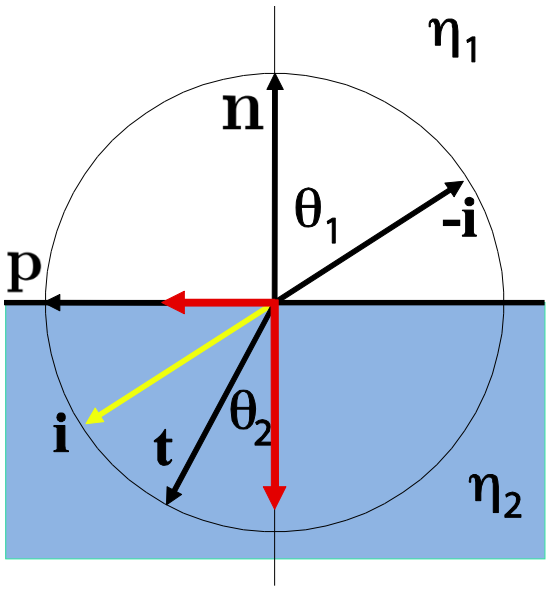


Find vectors in the refraction plane to express **t** with:

$$\mathbf{i_{perp}} = \mathbf{i} + \cos\Theta_1 \mathbf{n}$$

$$\mathbf{p} = \frac{\mathbf{i_{perp}}}{||\mathbf{i_{perp}}||}$$

Refraction



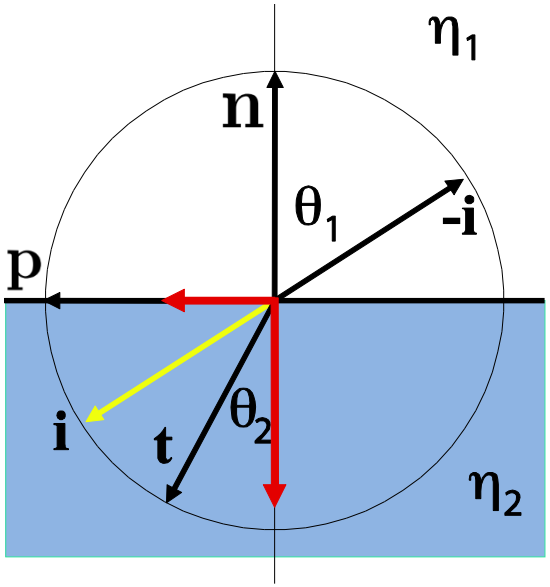
Snell's law:

$$\frac{\sin \Theta_2}{\sin \Theta_1} = \frac{\eta_1}{\eta_2} = \eta$$

We now know enough to solve **t** with

$$\mathbf{t} = - \cos \Theta_2 \mathbf{n} + \sin \Theta_2 \mathbf{p}$$

Refraction



Snell's law:

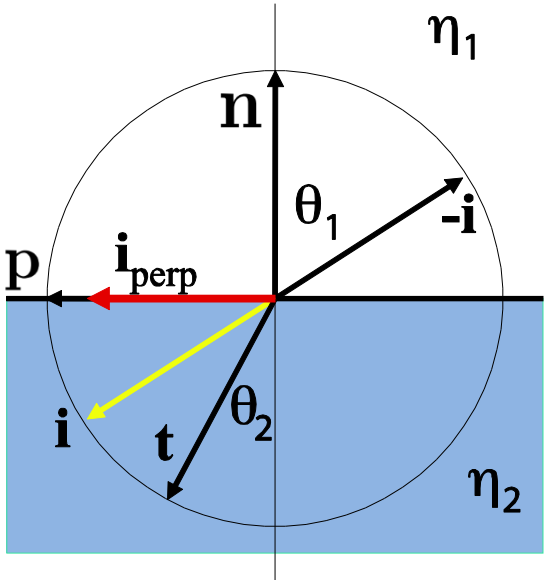
$$\frac{\sin\Theta_2}{\sin\Theta_1} = \frac{\eta_1}{\eta_2} = \eta$$

We now know enough to solve **t** with

$$\mathbf{t} = -\cos\Theta_2\mathbf{n} + \sin\Theta_2\mathbf{p}$$

We *could* stop here, but...

Refraction

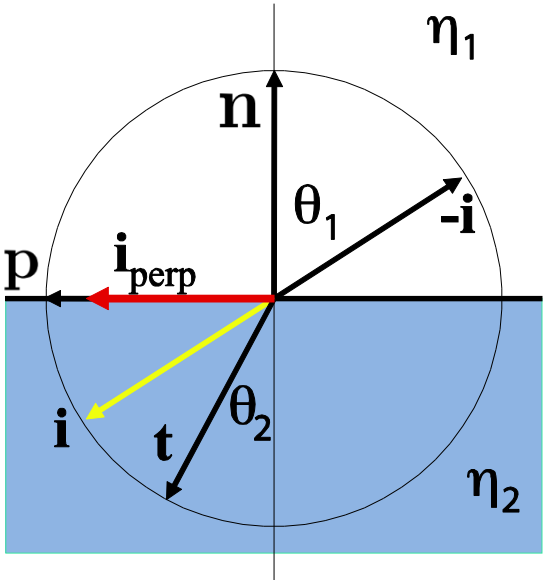


$$\mathbf{t} = - \cos\Theta_2\mathbf{n} + \sin\Theta_2\mathbf{p}$$

Simplification! *Recall…*

$$\mathbf{p} = \frac{\mathbf{i_{perp}}}{||\mathbf{i_{perp}}||}$$

Refraction



$$\mathbf{t} = - \cos\Theta_2\mathbf{n} + \sin\Theta_2\mathbf{p}$$

Simplification! *Recall…*

$$\mathbf{p} = \frac{\mathbf{i_{perp}}}{||\mathbf{i_{perp}}||}$$

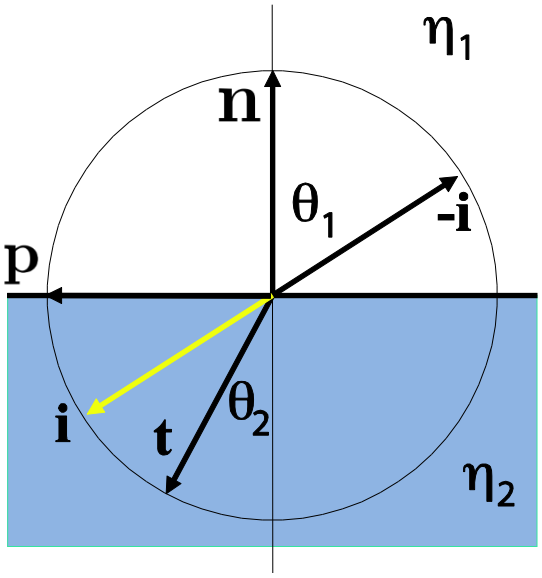
$$\mathbf{i_{perp}} = \mathbf{i} + \cos\Theta_1\mathbf{n}$$

$$||\mathbf{i_{perp}}|| = \sin\Theta_1$$

=>

$$\mathbf{t} = -\cos\Theta_2\mathbf{n} + \sin\Theta_2\frac{\mathbf{i}+\cos\Theta_1\mathbf{n}}{\sin\Theta_1}$$

Refraction



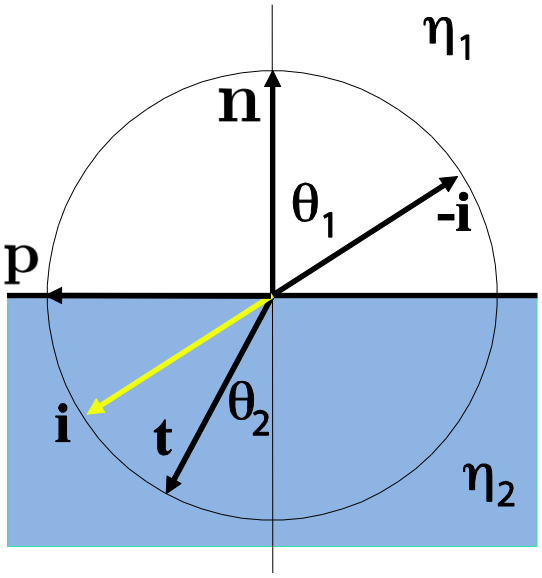
$$\mathbf{t} = -\cos\Theta_2\mathbf{n} + \sin\Theta_2\frac{\mathbf{i}+\cos\Theta_1\mathbf{n}}{\sin\Theta_1}$$

Snell again: $$\frac{\sin\Theta_2}{\sin\Theta_1} = \frac{\eta_1}{\eta_2} = \eta$$

=>

$$\mathbf{t} = \eta\mathbf{i} + (\eta\cos\Theta_1 - \cos\Theta_2)\mathbf{n}$$

Refraction



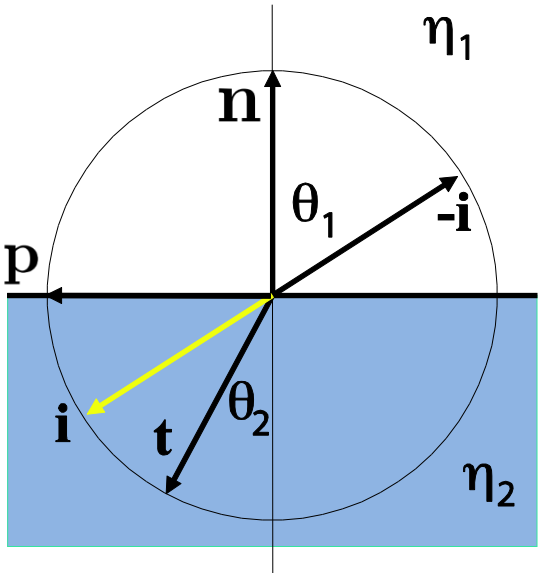
$$\mathbf{t} = -\cos\Theta_2\mathbf{n} + \sin\Theta_2\frac{\mathbf{i}+\cos\Theta_1\mathbf{n}}{\sin\Theta_1}$$

Snell again: $$\frac{\sin\Theta_2}{\sin\Theta_1} = \frac{\eta_1}{\eta_2} = \eta$$

=>

$$\mathbf{t} = \eta\mathbf{i} + (\eta\cos\Theta_1 - \cos\Theta_2)\mathbf{n}$$

Refraction



$$\mathbf{t} = -\cos\Theta_2\mathbf{n} + \sin\Theta_2\frac{\mathbf{i}+\cos\Theta_1\mathbf{n}}{\sin\Theta_1}$$

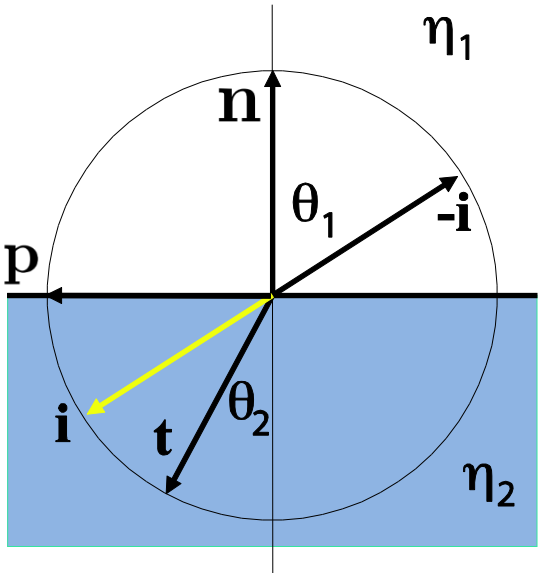
Snell again: $$\frac{\sin\Theta_2}{\sin\Theta_1} = \frac{\eta_1}{\eta_2} = \eta$$

=>

$$\mathbf{t} = \eta\mathbf{i} + (\eta\cos\Theta_1 - \cos\Theta_2)\mathbf{n}$$

Still need this one!

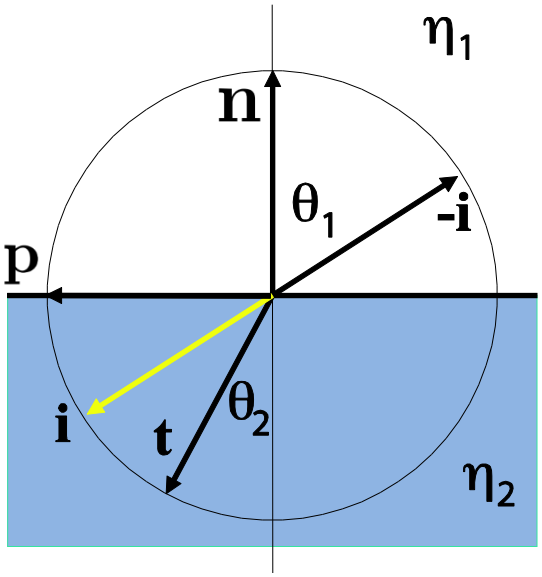
Refraction



$$\mathbf{t} = \eta\mathbf{i} + (\eta\cos\Theta_1 - \cos\Theta_2)\mathbf{n}$$

A familiar identity: $\cos^2\Theta_2 + \sin^2\Theta_2 = 1$

Refraction

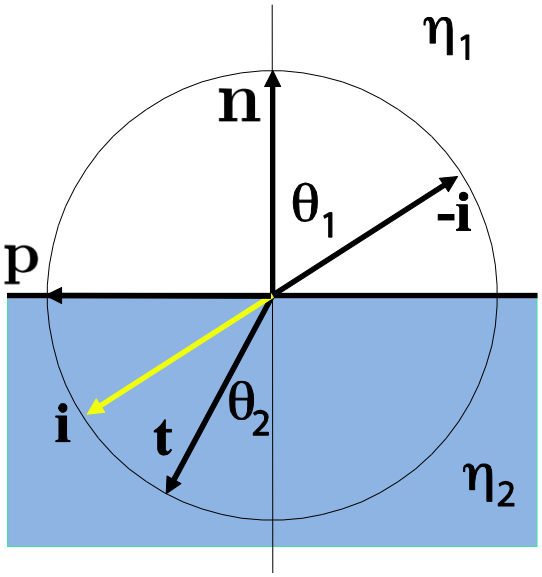


$$\mathbf{t} = \eta\mathbf{i} + (\eta \cos\Theta_1 - \cos\Theta_2)\mathbf{n}$$

A familiar identity: $\cos^2\Theta_2 + \sin^2\Theta_2 = 1$

$$\cos^2\Theta_2 = 1 - \frac{\sin^2\Theta_2 \sin^2\Theta_1}{\sin^2\Theta_1}$$

Refraction



$$\mathbf{t} = \eta\mathbf{i} + (\eta\cos\Theta_1 - \cos\Theta_2)\mathbf{n}$$

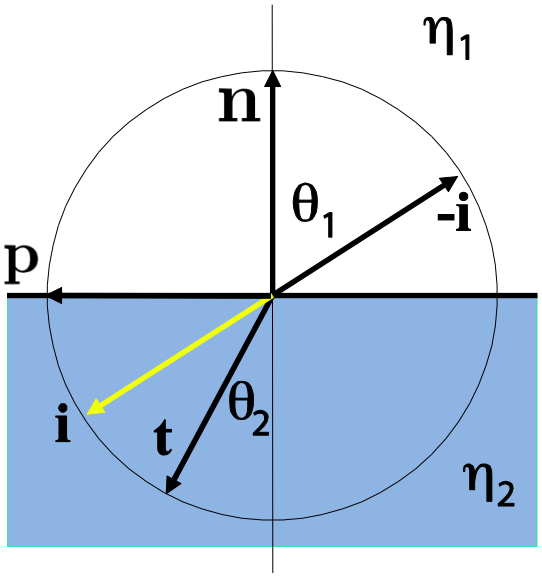
A familiar identity: $\cos^2\Theta_2 + \sin^2\Theta_2 = 1$

$$\cos^2\Theta_2 = 1 - \frac{\sin^2\Theta_2 \sin^2\Theta_1}{\sin^2\Theta_1}$$

$$\cos^2\Theta_2 = 1 - \frac{\sin^2\Theta_2(1-\cos^2\Theta_1)}{\sin^2\Theta_1}$$

Identity applied again

Refraction



$$\mathbf{t} = \eta\mathbf{i} + (\eta\cos\Theta_1 - \cos\Theta_2)\mathbf{n}$$

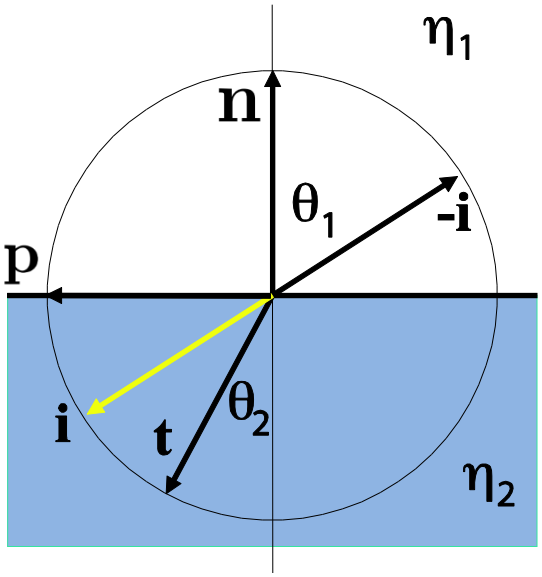
A familiar identity: $\cos^2\Theta_2 + \sin^2\Theta_2 = 1$

$$\cos^2\Theta_2 = 1 - \frac{\sin^2\Theta_2 \sin^2\Theta_1}{\sin^2\Theta_1}$$

$$\cos^2\Theta_2 = 1 - \frac{\sin^2\Theta_2(1-\cos^2\Theta_1)}{\sin^2\Theta_1}$$

Recognize this?

Refraction



$$\mathbf{t} = \eta\mathbf{i} + (\eta\cos\Theta_1 - \cos\Theta_2)\mathbf{n}$$

A familiar identity: $\cos^2\Theta_2 + \sin^2\Theta_2 = 1$

$$\cos^2\Theta_2 = 1 - \frac{\sin^2\Theta_2 \sin^2\Theta_1}{\sin^2\Theta_1}$$

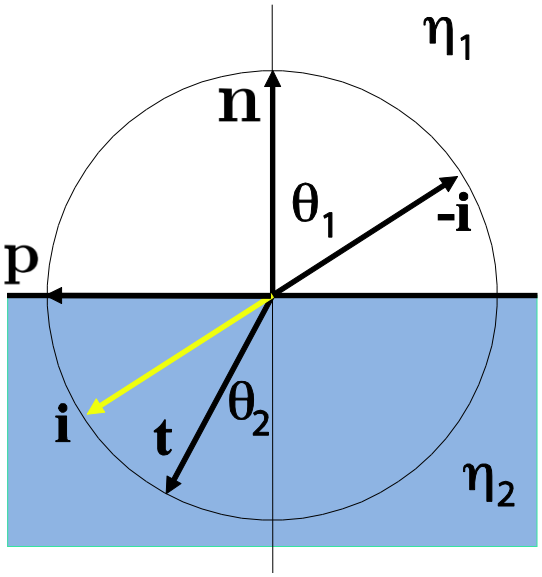
$$\cos^2\Theta_2 = 1 - \frac{\sin^2\Theta_2 (1-\cos^2\Theta_1)}{\sin^2\Theta_1}$$

$$\cos\Theta_2 = \sqrt{1 - \eta^2(1 - \cos^2\Theta_1)}$$

Hooray!

... and this one is easy: $\cos\Theta_1 = -\mathbf{i}\cdot\mathbf{n}$

Refraction



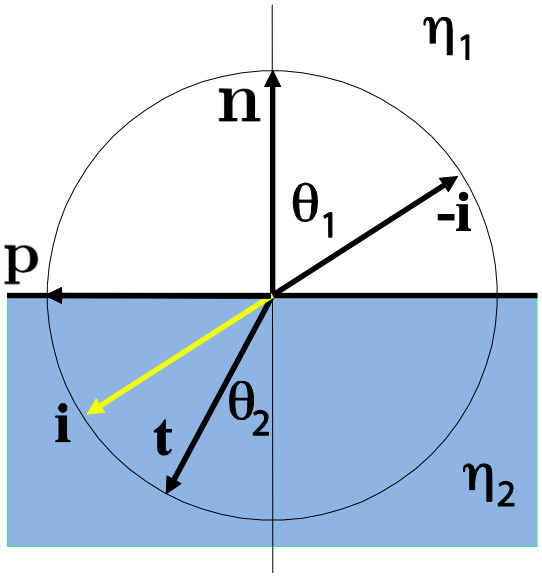
Tidying up…

$$r = -\,\mathbf{i}\cdot\mathbf{n}$$

$$c = 1 - \eta^2(1 - r^2)$$

$$\mathbf{t} = \eta\mathbf{i} + (\eta r - \sqrt{c})\mathbf{n}$$

Refraction



Finally…

$$r = -\,\mathbf{i}\cdot\mathbf{n}$$

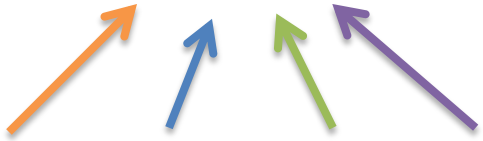
$$c = 1 - \eta^2(1 - r^2)$$

$$\mathbf{t} = \eta\mathbf{i} + (\eta r - \sqrt{c})\mathbf{n}$$

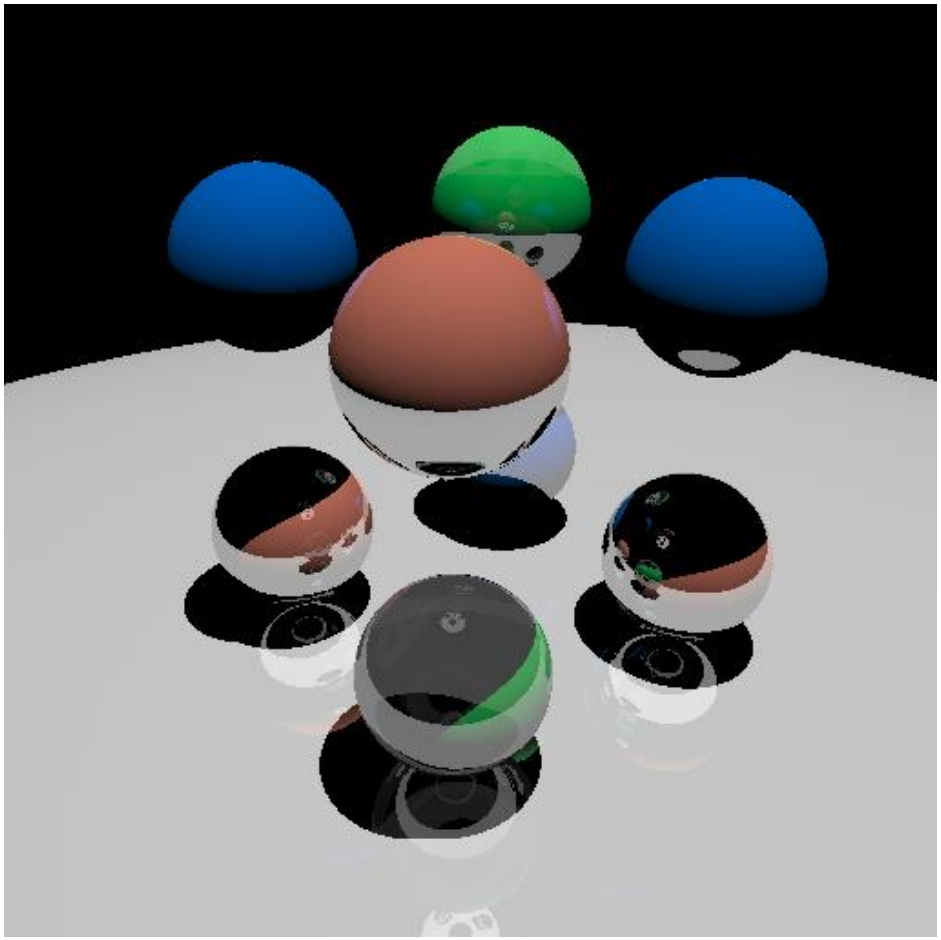
We can't solve for $\mathbf{c} < 0$

– Total internal reflection!

- Return ***reflection*** ray instead



Refraction

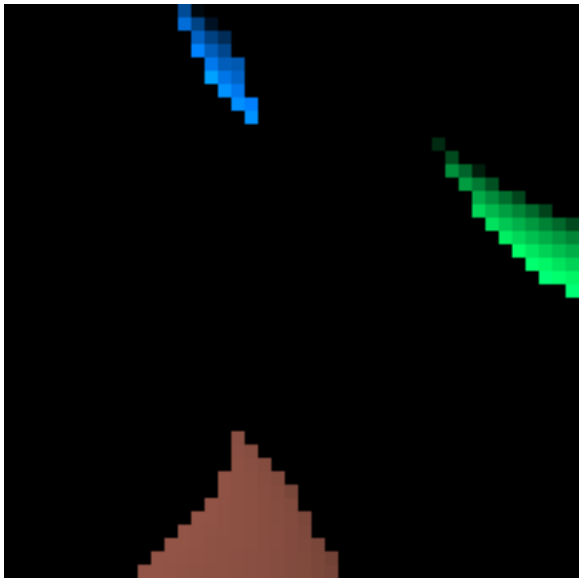


Assignment 1

- Diffuse Reflection
- Whitted ray tracing
  - Shadows
  - Reflections
  - Refractions
- Super sampling
- Ray-Triangle Intersection
- Blinn-Phong Shading *(Optional)*

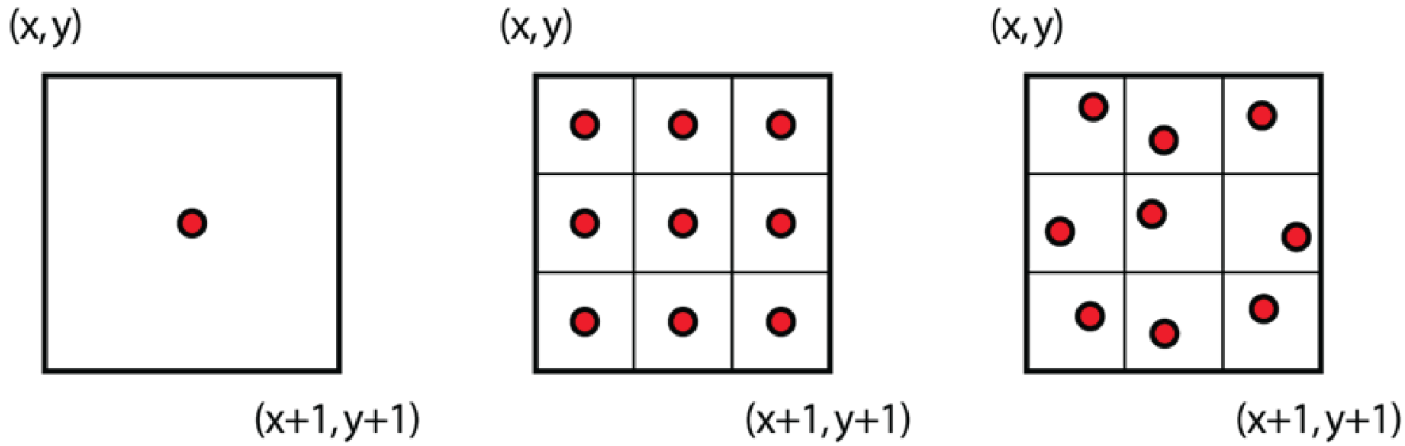
Super Sampling

The images look very jagged so far when we zoom in…



Super Sampling

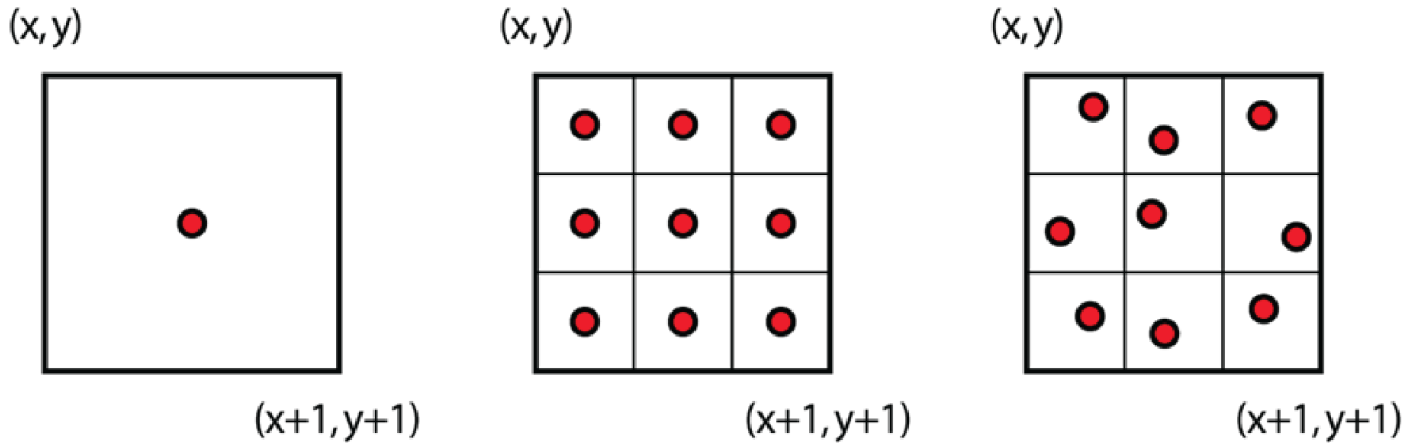
It’s because we only use 1 sample/pixel



Instead use NxN stratified samples

Super Sampling

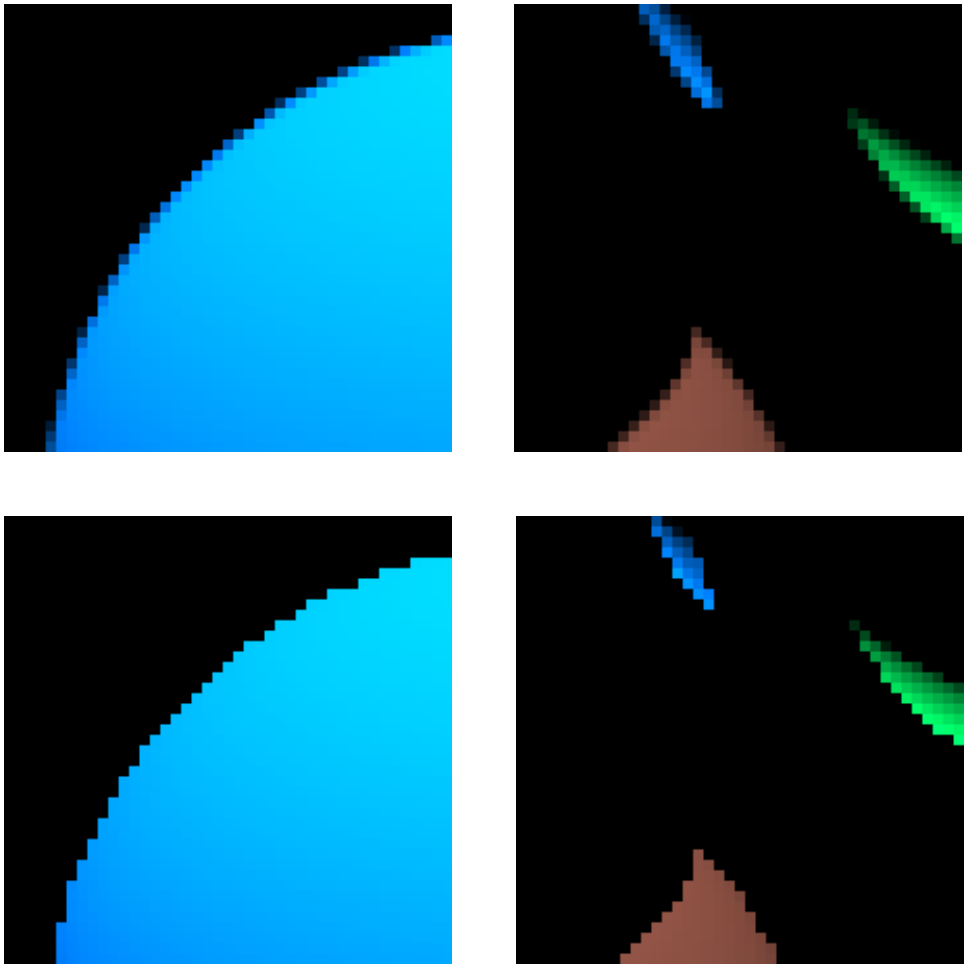
It’s because we only use 1 sample/pixel



Instead use NxN stratified samples

Hint: uniform() *returns a random value [0, 1)*

Super Sampling



Assignment 1

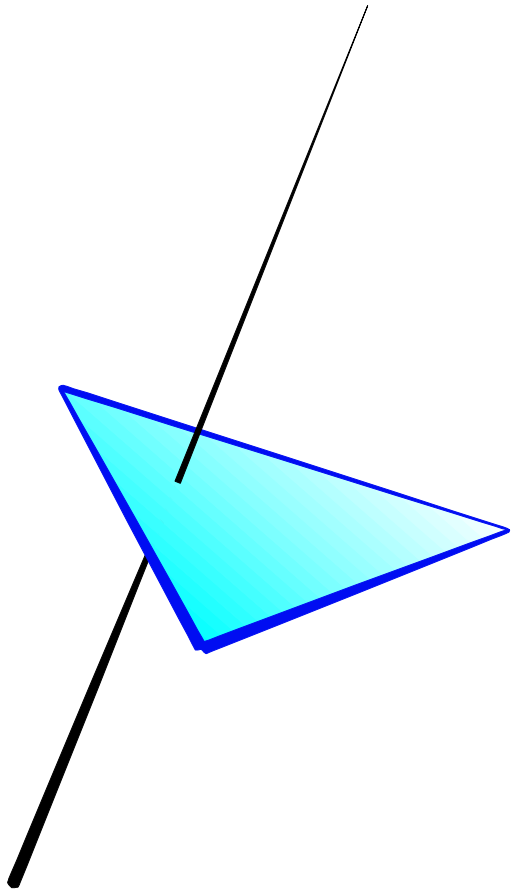
- Diffuse Reflection
- Whitted ray tracing
  - Shadows
  - Reflections
  - Refractions
- Super sampling
- Ray-Triangle Intersection
- Blinn-Phong Shading *(Optional)*

Important note!

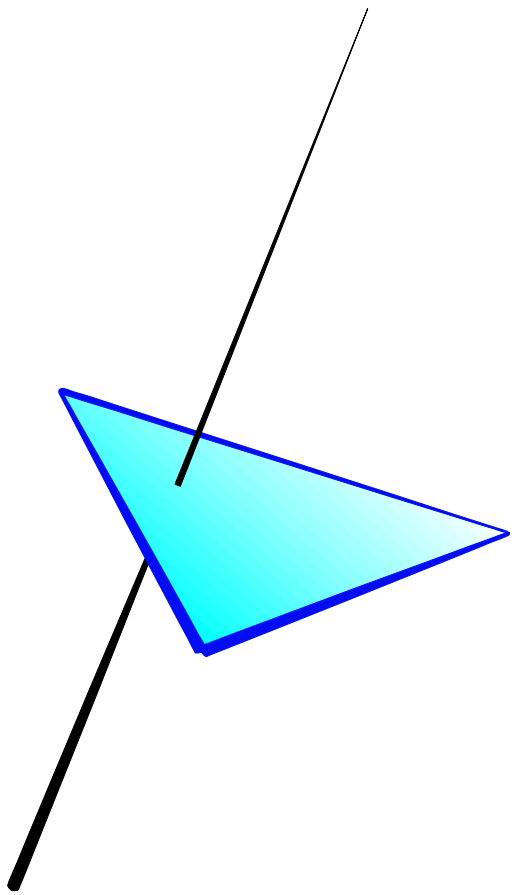
Before you start with the triangle intersection assignment:

- In the buildSpheres()-function:
  - Remove the big sphere (floor substitute)!
  - Otherwise the triangulated floor plane will be occluded!

Ray-Triangle Intersection



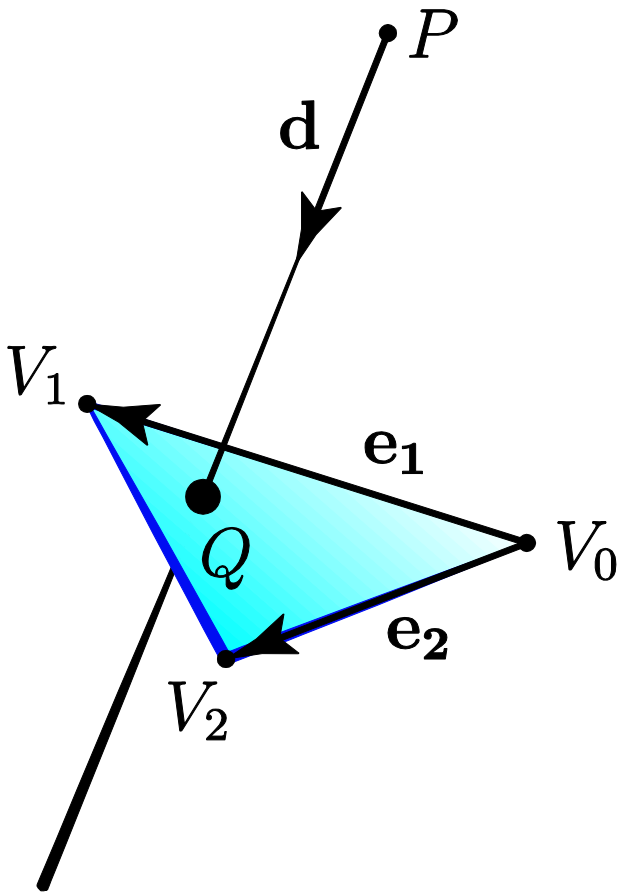
Ray-Triangle Intersection



Solve the intersection problem in two steps:

1. Find the intersection point *(Q)* of the ray with the triangle plane
2. Determine if *Q* is inside the triangle bounds using barycentric coordinates

Ray-Triangle Intersection



Recall…

A triangle is defined by three vertices

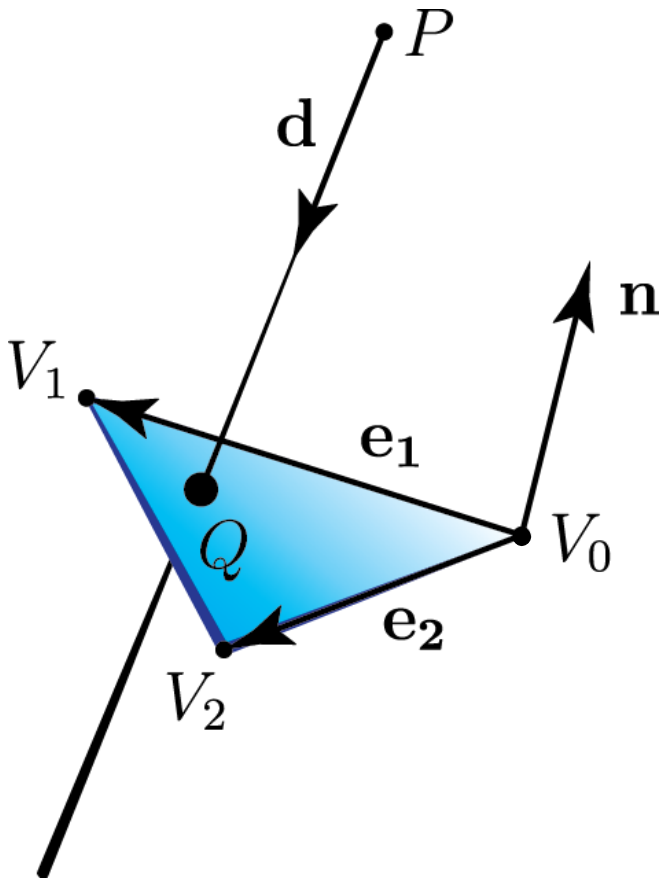
V_i , i = {0, 1, 2}

A ray is defined by some origin *P*, and a direction vector **d**.

An intersection with the triangle plane will occur at some distance *t* along the ray.

$Q = P + t\mathbf{d}$

Ray-Triangle Intersection



Step 1: Find Q

Plane normal

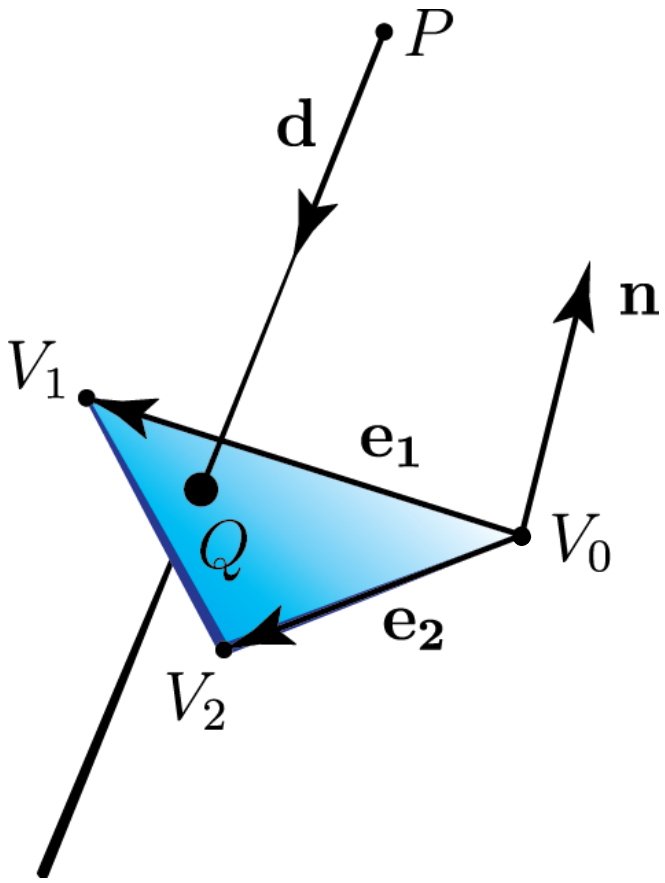
$$\mathbf{n} = \mathbf{e}_1 \times \mathbf{e}_2$$

Plane equation

$$\mathbf{n} \cdot X + m = 0$$
$$m = -\mathbf{n} \cdot V_0$$

Note: The magnitude of **n** corresponds to 2x the area of the triangle.

Ray-Triangle Intersection



Step 1: Find Q

Plane intersection

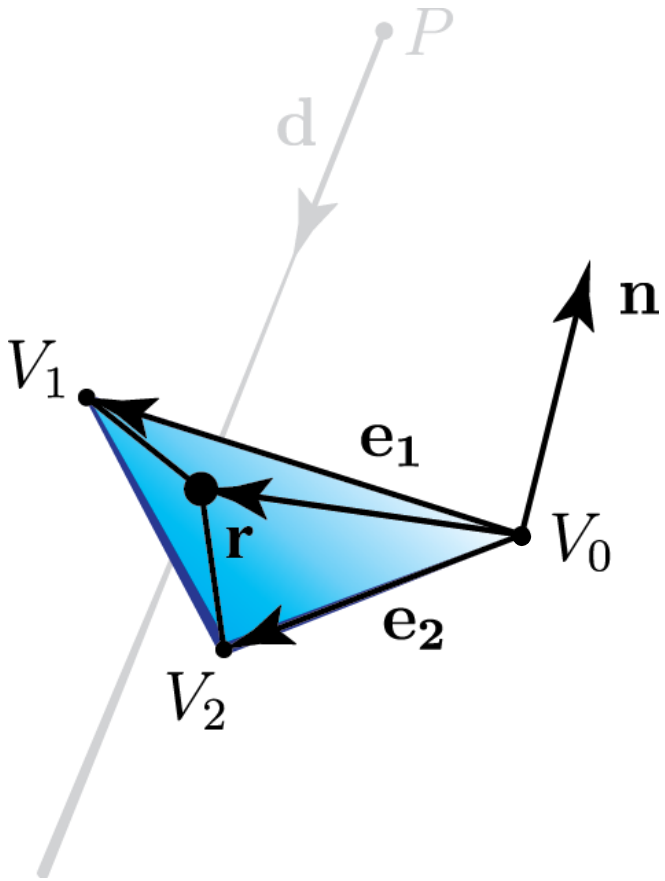
$$t = \frac{\mathbf{n} \cdot P + m}{\mathbf{n} \cdot \mathbf{d}}$$

$$Q = P + t\mathbf{d}$$

Must also make sure that

$$t_{min} < t < t_{max}$$

Ray-Triangle Intersection



Step 2: Find barycentric coordinates

Create vector **r**, which is *coplanar* with $\mathbf{e}_1$ and $\mathbf{e}_2$

$$\mathbf{r} = Q - V_0$$

Barycentric *v* and *w*

$$v = \frac{||\mathbf{e}_1 \times \mathbf{r}||}{||\mathbf{n}||} \quad w = \frac{||\mathbf{r} \times \mathbf{e}_2||}{||\mathbf{n}||}$$

Barycentric coordinates can be expressed as the area of a "sub-triangle" divided by the area of the whole triangle

Thus, if the following holds, *Q* is inside the triangle

$$v \geq 0$$
$$w \geq 0$$
$$v + w < 1$$

Ray-Triangle Intersection

The method presented here is *not* the fastest one available. Many optimizations are possible.

A faster test can be found here:

http://www.cs.virginia.edu/~gfx/Courses/2003/ImageSynthesis/papers/Acceleration/Fast%20MinimumStorage%20RayTriangle%20Intersection.pdf

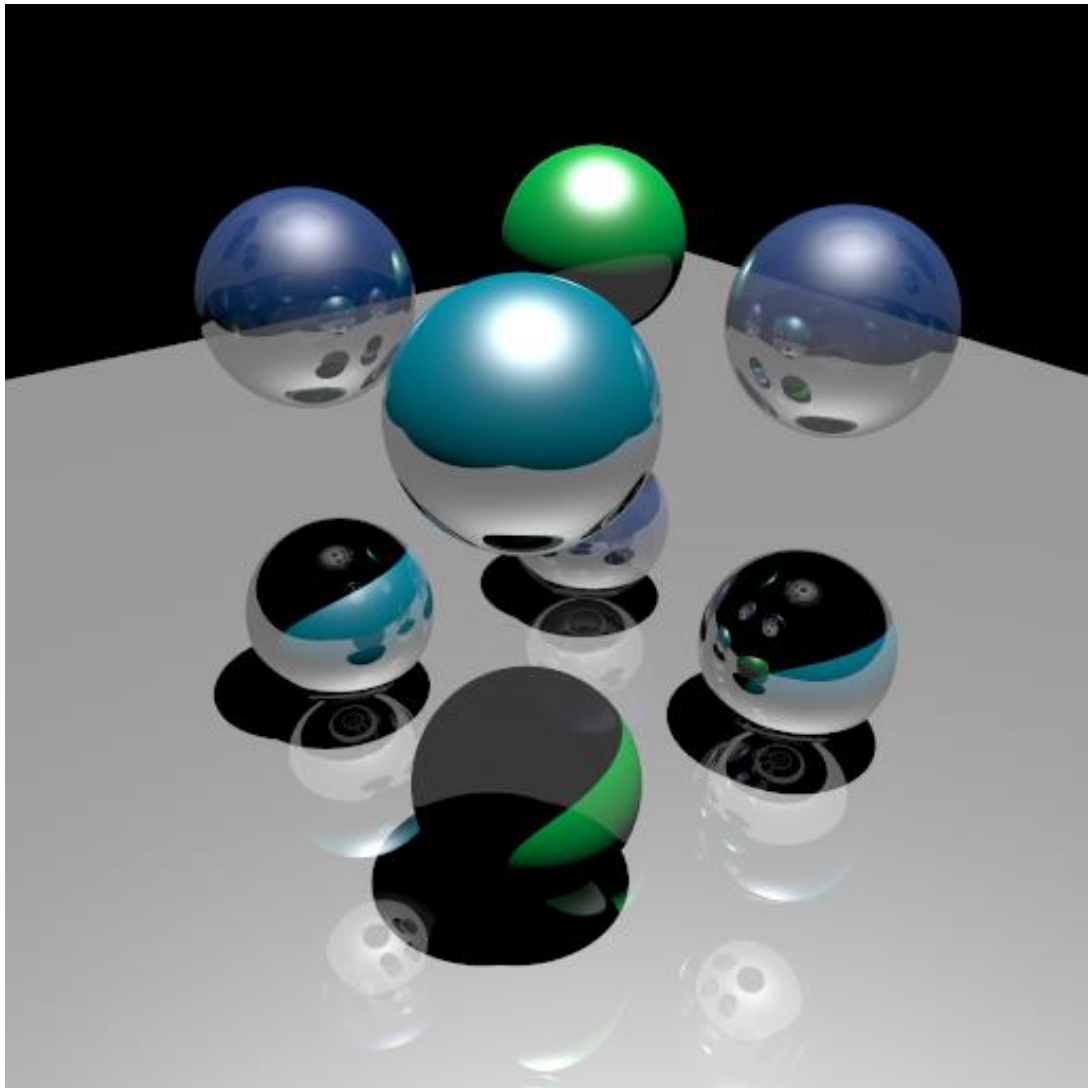
(or just google ”Fast Minimum Storage Ray Triangle Intersection”)

…or go to the end of this slide deck.

Assignment 1

- Diffuse Reflection
- Whitted ray tracing
  - Shadows
  - Reflections
  - Refractions
- Super sampling
- Ray-Triangle Intersection
- Blinn-Phong Shading *(Optional)*

Everything put together



Running on the Lab Machines

Debugging the code:

- Run in debug mode
- Use breakpoints and look at variables
- Still not working? The forum is a good place to find answers and ask questions

Hot tip x2:

- Debug mode is slow, Release is fast!
- Render at low resolutions (~64 x 64) for debugging!

That’s all for assignment 1…

Let’s talk about BRDFs!

Recall the radiance calculation

$$L_{out}(x \rightarrow \Theta) = L_{in}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi)$$

- Incoming radiance — light->getRadiance();
- **BRDF** — **is.mMaterial.evalBRDF(is,Ψ);**
- Incident angle — max(Ψ.dot(is.mNormal), 0.0f);

Bidirectional **R**eflectance **D**istribution **F**unction

The BRDF describes the radiance to irradiance ratio for all combinations of irradiance and radiance directions (**Ψ** and **ϴ**) at a surface point *x*.

$$f_r(x, \Psi \leftrightarrow \Theta)$$

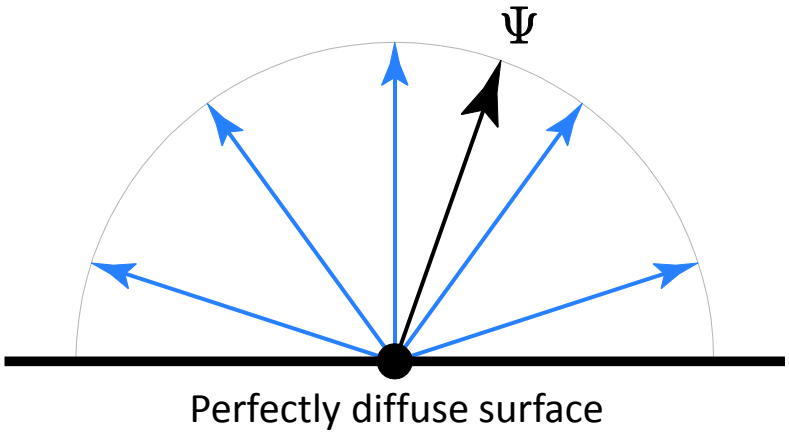
BRDF

Commonly modeled features

- Multiple layers
- Geometrical attenuation
- Micro-facets
- Fresnel term
- Anisotropy

BRDF

Diffuse Reflection

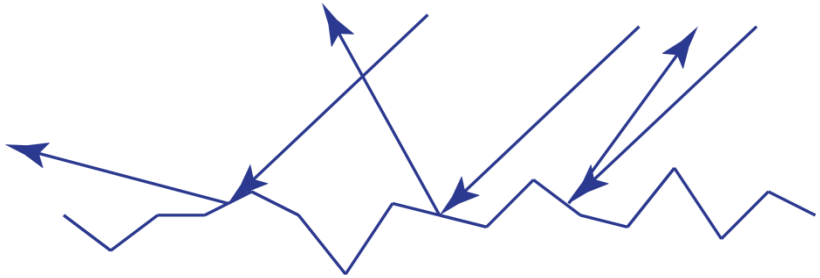


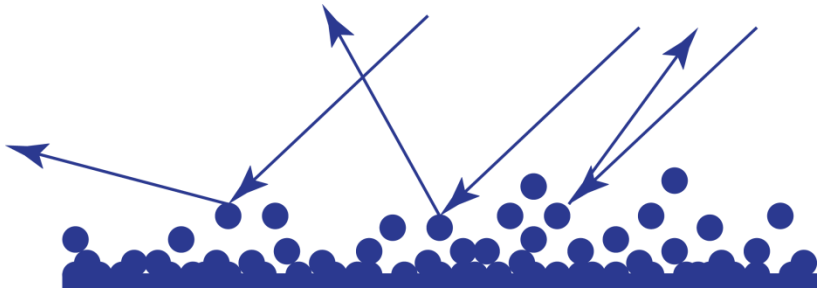
Light is redistributed equally in all directions ⇔
BRDF is constant: $\mathbf{k_d}$

(aka Lambert shading)

BRDF

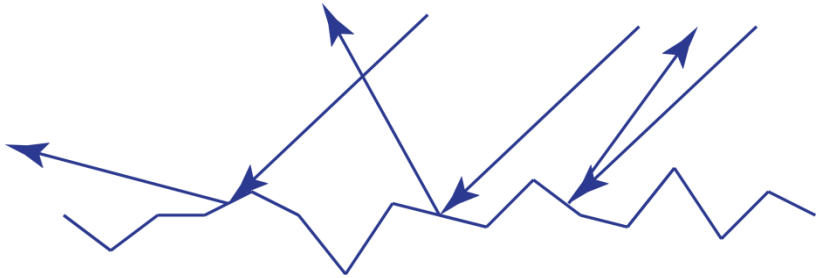
Diffuse Reflection

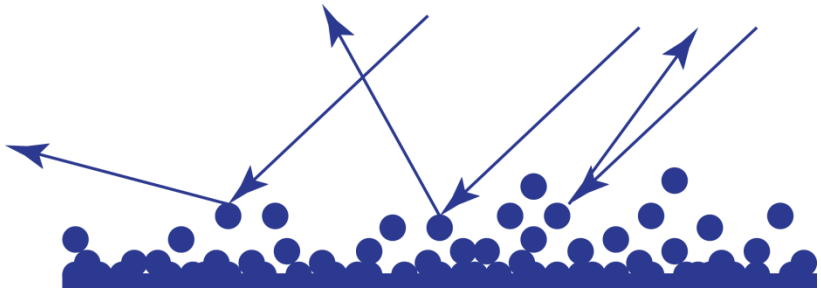




BRDF

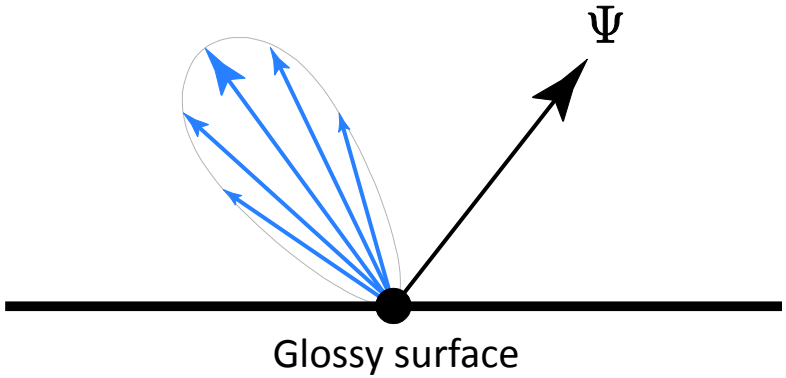
Diffuse Reflection





BRDF

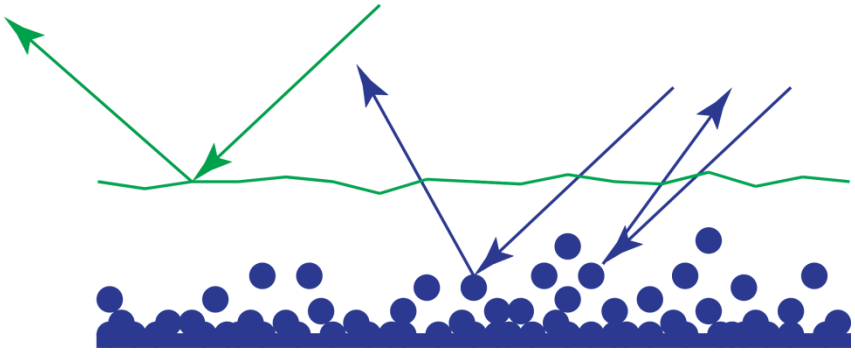
Specular Glossy Reflection



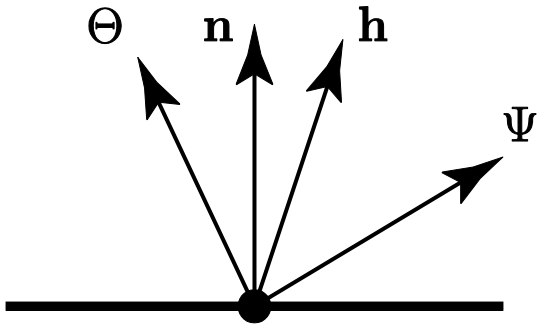
Light is redistributed in the general reflection direction.

BRDF

Diffuse + Specular Glossy Reflection



Blinn-Phong BRDF

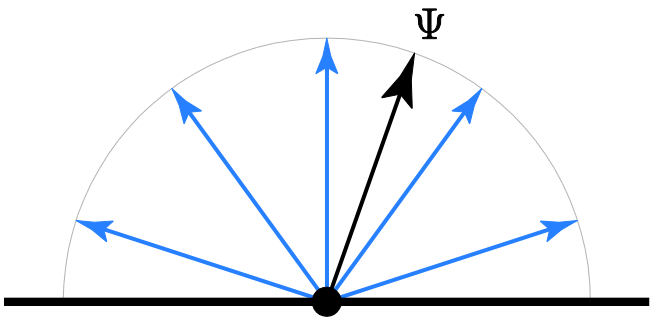


$$\mathbf{h} = \frac{\boldsymbol{\Psi}+\boldsymbol{\Theta}}{|\boldsymbol{\Psi}+\boldsymbol{\Theta}|}$$

$$\text{Diffuse component} = \frac{\rho_d}{\pi}$$

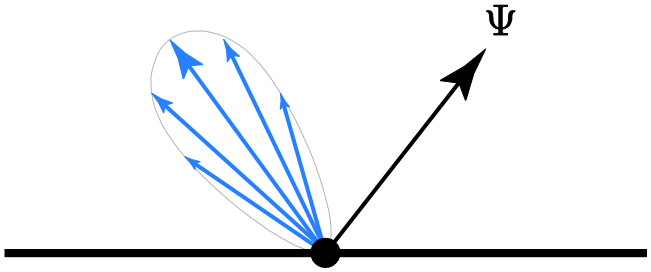
$$\text{Specular component} = \rho_s \frac{s+4}{8\pi}(\mathbf{h}\cdot\mathbf{n})^s$$

Result = **Diffuse** + **Specular**



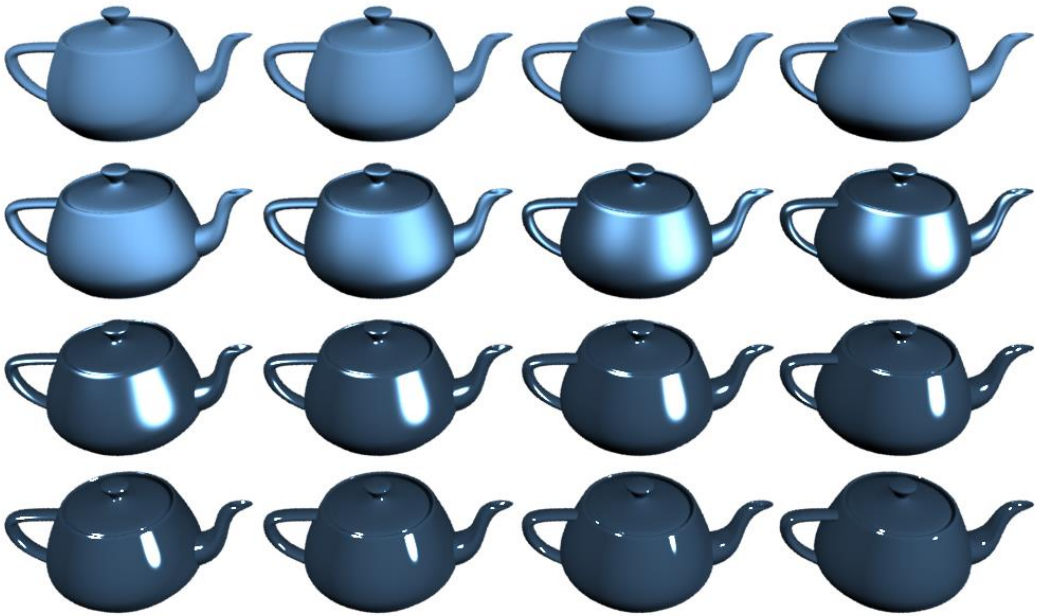
Diffuse

+



Specular (Glossy)

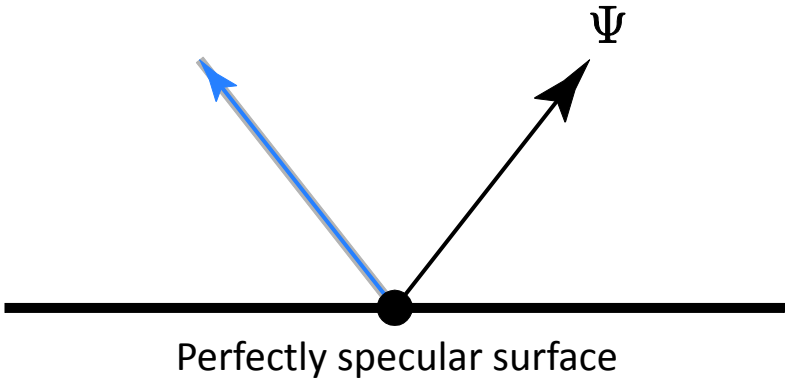
Blinn-Phong BRDF



Incr. shininess: **s**
(roughness)

Blinn-Phong & Whitted reflection

Perfect Specular Reflection



Mirror surface – Light is redistributed in the exact reflection direction only.

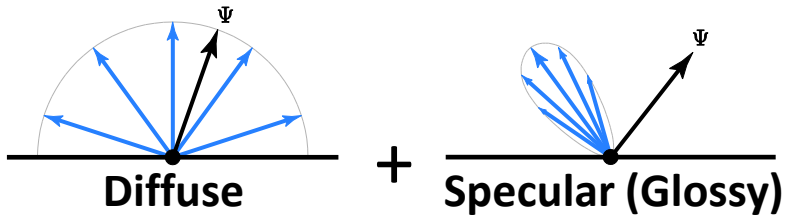
Blinn-Phong & Whitted reflection

Incoming light + reflection

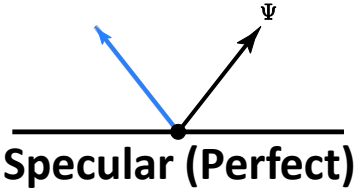
$$L_{out}(x \rightarrow \Theta) = (1 - r) \sum_{i}^{lights} L_i(x \leftarrow \Psi_i) f_r(x, \Psi_i \leftrightarrow \Theta) cos(N_x, \Psi_i) + r \; L_s(x \rightarrow R)$$

r = *Specular reflectance parameter*

BRDF (Blinn-Phong) $f_r(x, \Psi_i \leftrightarrow \Theta)$ =



Reflection $L_s(x \rightarrow R)$ =



BRDF

There are *lots* of different BRDFs for different purposes…

- Material
- Realism vs. aesthetics
- Computational cost vs. quality
- Flexibility
- …

Example: Schlick's BRDF

Commonly modeled features

☑ Multiple layers

☑ Geometrical attenuation

- *(But I won't cover it here)*

☑ Fresnel term

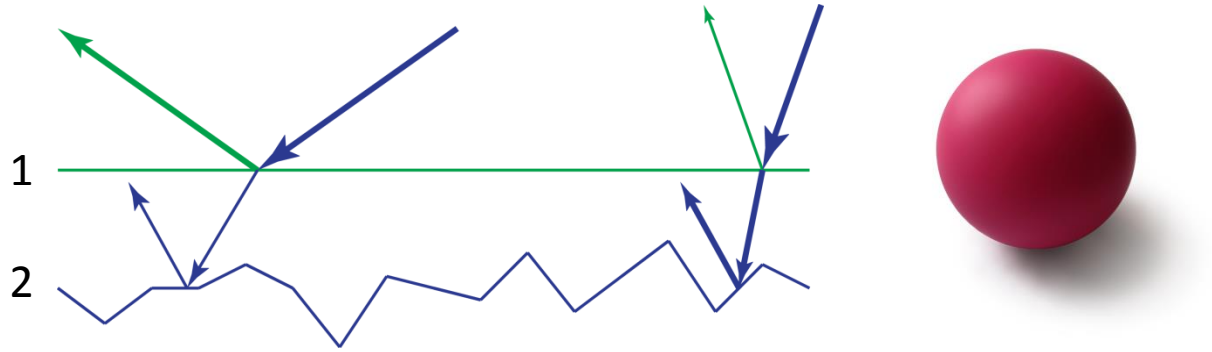
☑ Anisotropy

\+ Diffuse ⇔ Specular continuum

Schlick's BRDF

Multi-layered approach

$$f_r(x, \Psi \leftrightarrow \Theta) = S_1 D_1(x, \Psi, \Theta) + (1 - S_1) S_2 D_2(x, \Psi, \Theta)$$



Use one or two layers.

Layer contribution is dependent on incident angle.

Schlick's BRDF

Multi-layered approach

$$f_r(x, \Psi \leftrightarrow \Theta) = S_1 D_1(x, \Psi, \Theta) + (1 - S_1) S_2 D_2(x, \Psi, \Theta)$$

Use either

Constant

("Phong-like")

$$S_i = C_i$$

or

Incident angle dependent

(Fresnel approximation)

$$\mathbf{h} = \frac{\mathbf{\Psi} + \mathbf{\Theta}}{|\mathbf{\Psi} + \mathbf{\Theta}|}$$

$$u = \mathbf{h} \cdot \mathbf{\Theta}$$

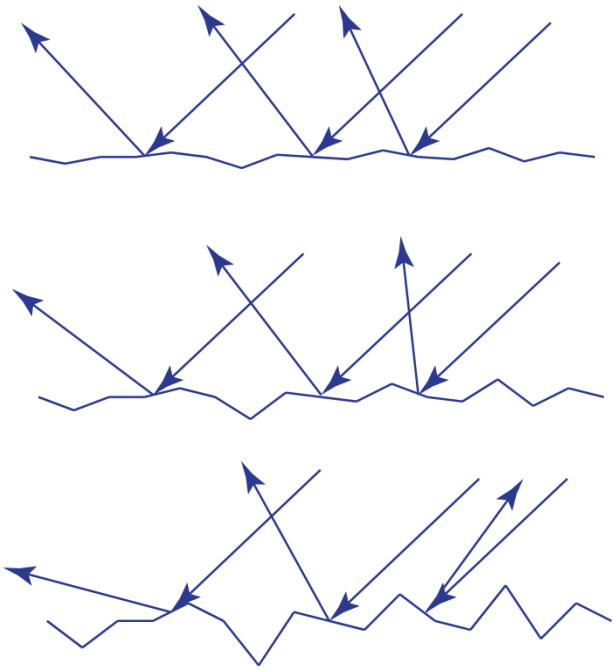
$$S_i = C_i + (1 - C_i)(1 - u)^5$$

Reflection factor: $\mathbf{C_i}$

Schlick's BRDF

Diffuse ⇔ Specular continuum

Roughness factor: **r**



Good for metals, for example

Schlick's BRDF

Diffuse ⇔ Specular continuum

$$f_r(x, \Psi \leftrightarrow \Theta) = S_1 D_1(x, \Psi, \Theta) + (1 - S_1) S_2 D_2(x, \Psi, \Theta)$$

$$D_i = AZ$$

Zenith:

$$m = \mathbf{h} \cdot \mathbf{n}$$

$$Z = \frac{r}{(1 + rm^2 - m^2)^2}$$

Roughness factor: **r**

Z goes from perfectly diffuse (r = 1) to perfectly specular (r = 0)

Schlick's BRDF

Anisotropy

$$f_r(x, \Psi \leftrightarrow \Theta) = S_1 D_1(x, \Psi, \Theta) + (1 - S_1) S_2 D_2(x, \Psi, \Theta)$$

$$D_i = AZ$$

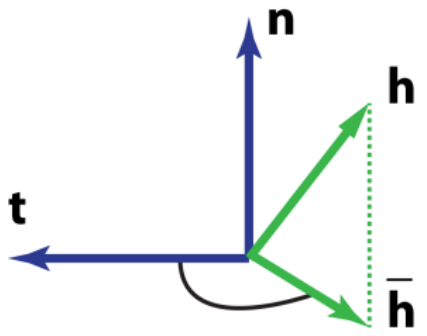
Azimuth:

$$\overline{\mathbf{h}} = \mathbf{h} - \operatorname{proj}_{\mathbf{n}} \mathbf{h}$$

$$w = \mathbf{t} \cdot \overline{\mathbf{h}}_{\text{normalized}}$$

$$A = \sqrt{\frac{p}{p^2 - p^2 w^2 + w^2}}$$

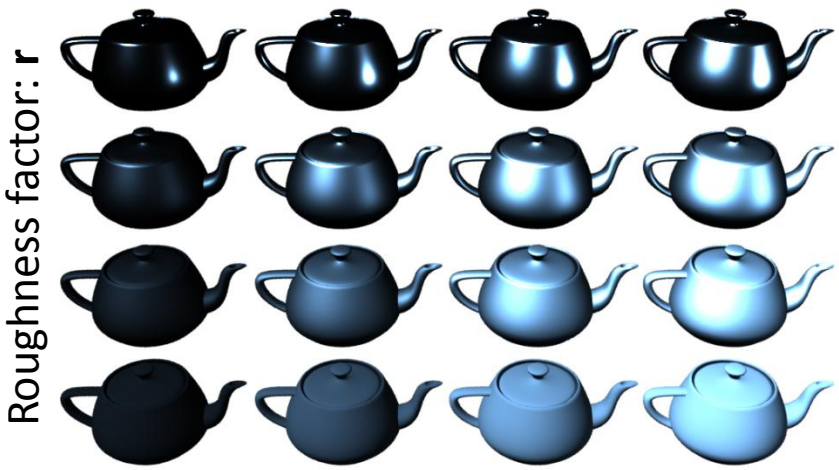
Isotropy factor: **p**

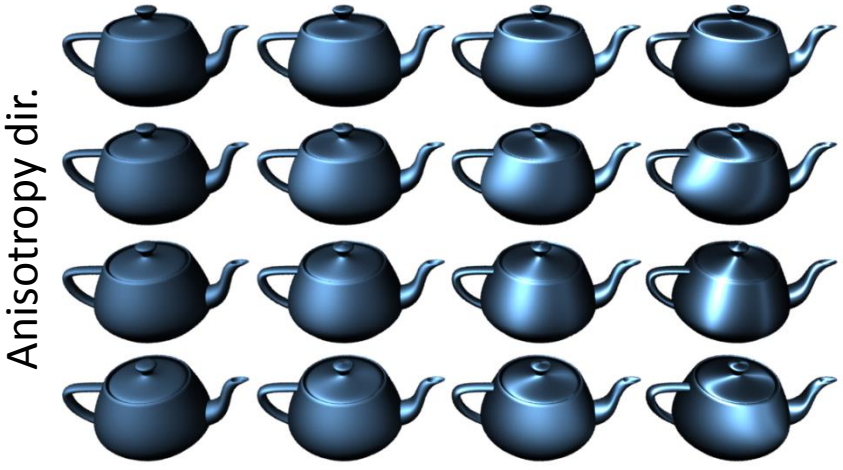


A goes from isotropic (p = 1) to anisotropic (p = 0)

Schlick's BRDF

(Single layered)



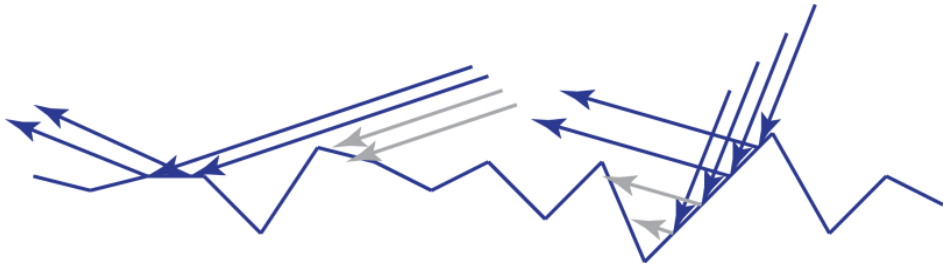


For more details, take a look at the original paper

http://igorsklyar.com/system/documents/papers/28/Schlick94.pdf

BRDF

Geometrical factor (Cook-Torrance)



Ratio of light that is *not* obstructed by the surface itself

BRDF

Geometrical factor (Cook-Torrance)

$$G_{in} = \frac{2(\mathbf{n}\cdot\mathbf{h})(\mathbf{n}\cdot\mathbf{\Theta})}{(\mathbf{h}\cdot\mathbf{\Theta})}$$

$$G_{out} = \frac{2(\mathbf{n}\cdot\mathbf{h})(\mathbf{n}\cdot\mathbf{\Psi})}{(\mathbf{h}\cdot\mathbf{\Theta})}$$

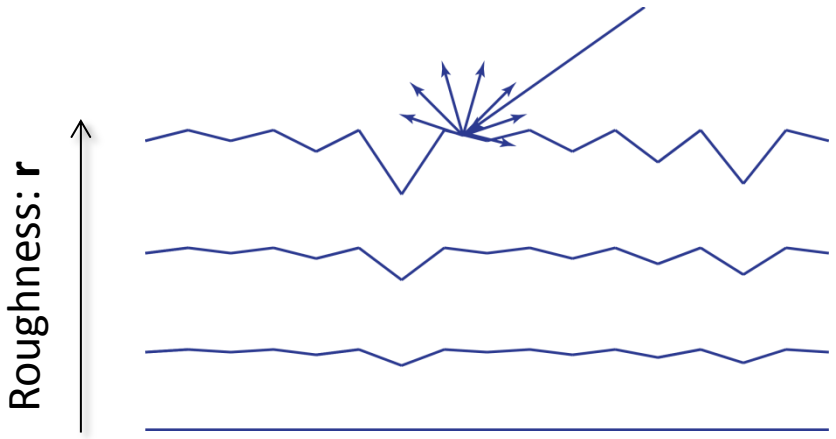
$$G = \min(G_{in}, G_{out}, 1)$$

Accounts for micro-facet occlusion.

Each facet is treated as a *mirror.*

http://www.cs.columbia.edu/~belhumeur/courses/appearance/cook-torrance.pdf

Oren-Nayar BRDF



Treats each micro-facet as a *diffuse* surface.

Roughness = 0 ⇔ Lambert

Oren-Nayar BRDF

$$f_r(x, \Psi \leftrightarrow \Theta) = A + B\gamma \sin\alpha \tan\beta$$

$$A = 1 - \frac{r^2}{2(r^2+0.33)} \qquad B = \frac{0.45r^2}{r^2+0.09}$$

$$\alpha = \max(\mathbf{n}\cdot\mathbf{\Psi}, \mathbf{n}\cdot\mathbf{\Theta}) \qquad \beta = \min(\mathbf{n}\cdot\mathbf{\Psi}, \mathbf{n}\cdot\mathbf{\Theta})$$

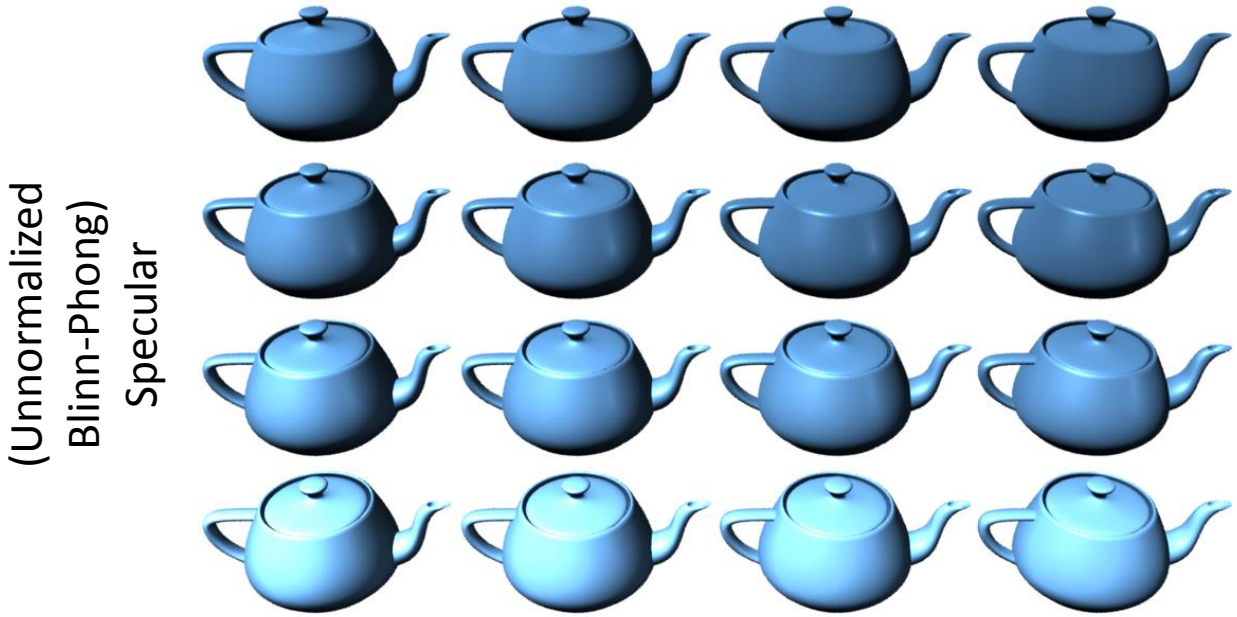
$$\gamma = \max(0, \cos(\phi_\Psi - \phi_\Theta))$$

Roughness: **r**

Treats each micro-facet as a *diffuse* surface.

Roughness = 0 ⇔ Lambert

Oren-Nayar BRDF



Original paper:

http://www1.cs.columbia.edu/CAVE/publications/pdfs/Oren_SIGGRAPH94.pdf

BRDF

Perfect subject for the final project!

We have now briefly looked at a few… but many more exist!
Some popular ones include:

- Blinn-Phong
- Schlick
- Oren-Nayar
- Cook-Torrence
- Ward
- Ashikhmin-Shirley
- Lafortune
- …

http://digibug.ugr.es/bitstream/10481/19751/1/rmontes_LSI-2012-001TR.pdf

Adding new BRDFs to prTracer

$$L_{out}(x \rightarrow \Theta) = L_{in}(x \leftarrow \Psi) f_r(x, \Psi \leftrightarrow \Theta) cos(N_x, \Psi)$$

- Incoming radiance — light->getRadiance();
- **BRDF** — **is.mMaterial.evalBRDF(is, Ψ);**
- Incident angle — max(Ψ.dot(is.mNormal), 0.0f);

Create new sub-class of <<Material>>

Implement your own evalBRDF-function

For OBJ files: Set up new material in Mesh::CreateMaterial()

That is all.

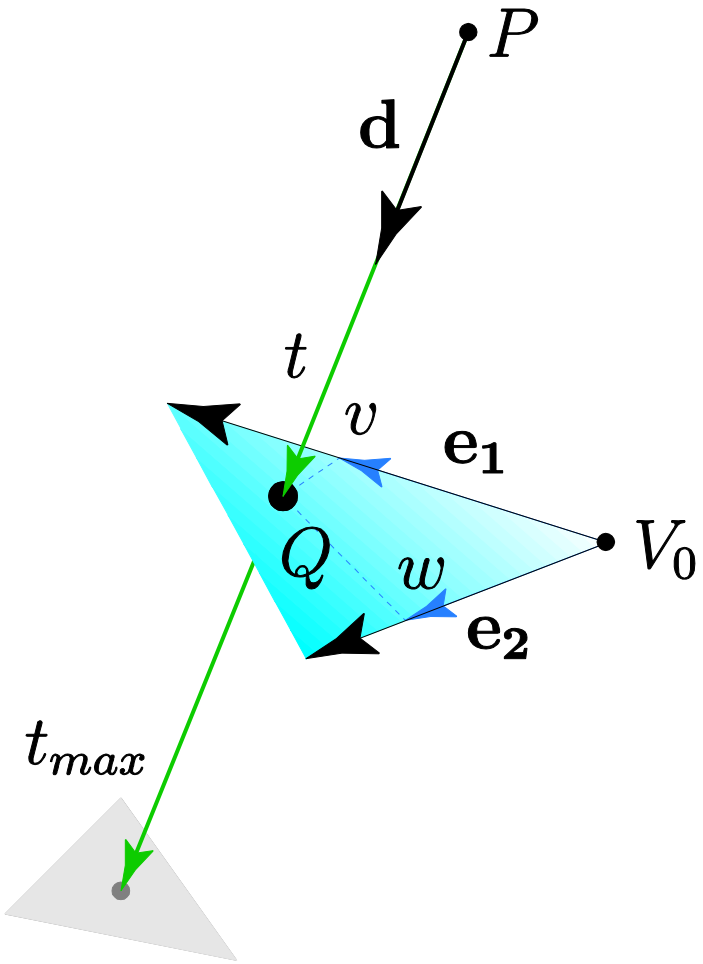
The first assignment is out now!

We'll be active on the forum, so be sure to check in if you have any comments or questions!

Fin

More optimized

Ray-Triangle Intersection



Q is inside the triangle if

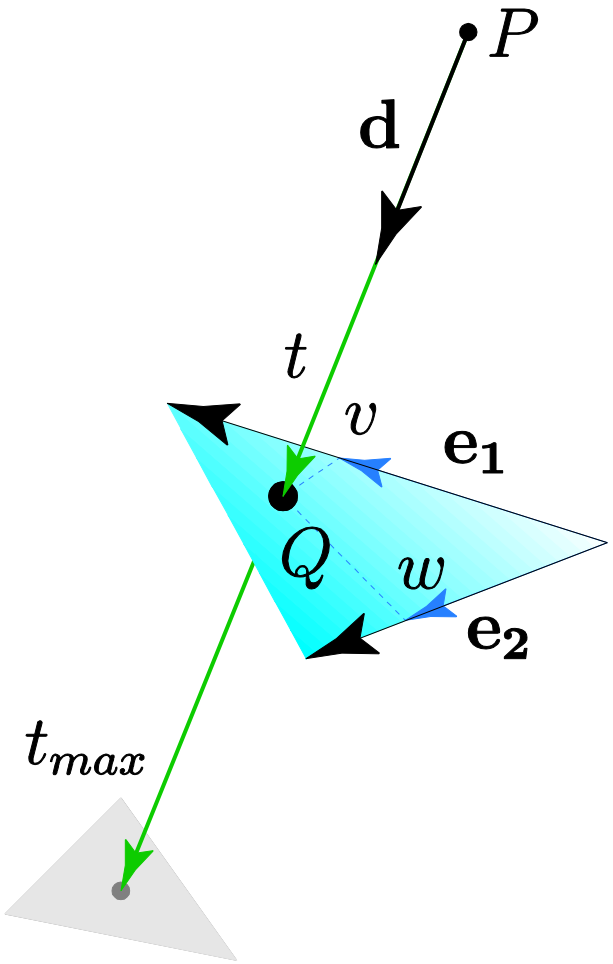
$$Q = V_0 + v\mathbf{e}_1 + w\mathbf{e}_2$$
$$v \geq 0$$
$$w \geq 0$$
$$v + w \leq 1$$

Must also make sure that

$$t_{min} < t < t_{max}$$

More optimized

Ray-Triangle Intersection

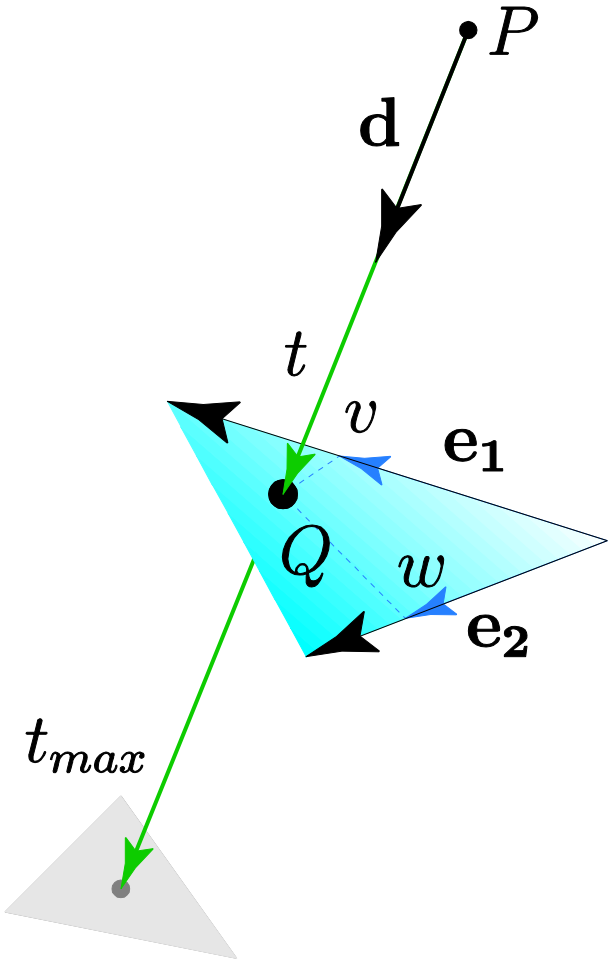


Substitute *Q* with *ray* and solve for *v, w* and *t*

$$P + t\mathbf{d} = V_0 + v\mathbf{e}_1 + w\mathbf{e}_2$$

More optimized

Ray-Triangle Intersection



Gather *v, w* and *t* to one side

$$P + t\mathbf{d} = V_0 + v\mathbf{e}_1 + w\mathbf{e}_2$$

=>

$$-t\mathbf{d} + v\mathbf{e}_1 + w\mathbf{e}_2 = P - V_0$$

=>

$$\begin{pmatrix} -d^x & e_1^x & e_2^x \\ -d^y & e_1^y & e_2^y \\ -d^z & e_1^y & e_2^z \end{pmatrix} \begin{pmatrix} t \\ v \\ w \end{pmatrix} = P - V_0 = \mathbf{r}$$

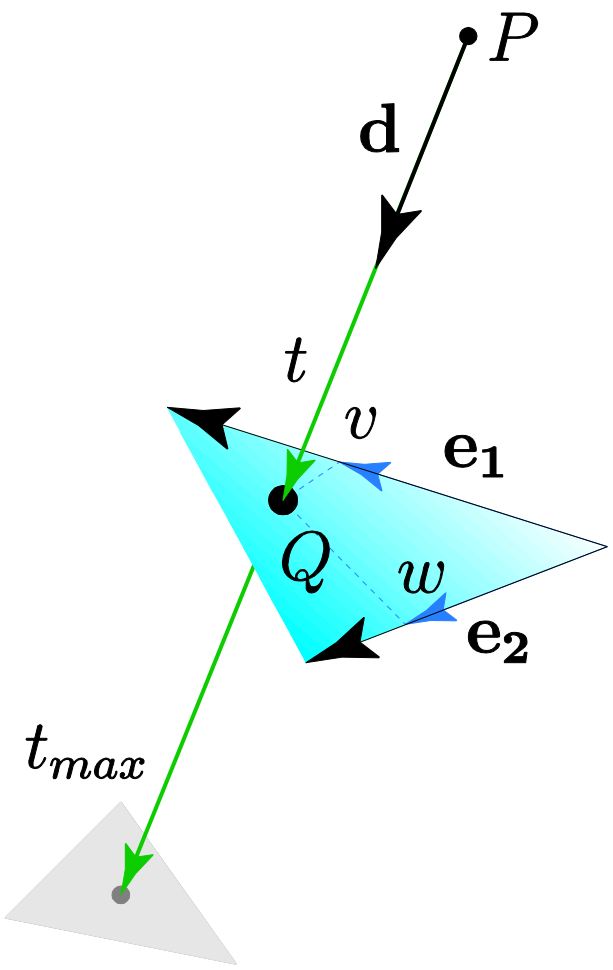
Neat, a 3x3 system!

Solve with Cramer's Rule

(Note: Not the same **r** *as in previous tri-ray slides!)*

More optimized

Ray-Triangle Intersection



Start by calculating the denominator *s*, which is the determinant of the 3x3 matrix

$$s = -\mathbf{d} \cdot (\mathbf{e}_1 \times \mathbf{e}_2)$$

Next we solve for *v*, *w*, *t*

$$\begin{aligned} t &= \mathbf{r} \cdot (\mathbf{e}_1 \times \mathbf{e}_2) / s \\ v &= -\mathbf{d} \cdot (\mathbf{r} \times \mathbf{e}_2) / s \\ w &= -\mathbf{d} \cdot (\mathbf{e}_1 \times \mathbf{r}) / s \end{aligned}$$

Optimize!

- Remove common sub expressions
- Remember:
  - a . (b x c) = c . (a x b) = -c . (b x a)
- If you've done it right: 1 division, 2 cross products, 4 dot products and a bunch of *if*s and cheaper ops.