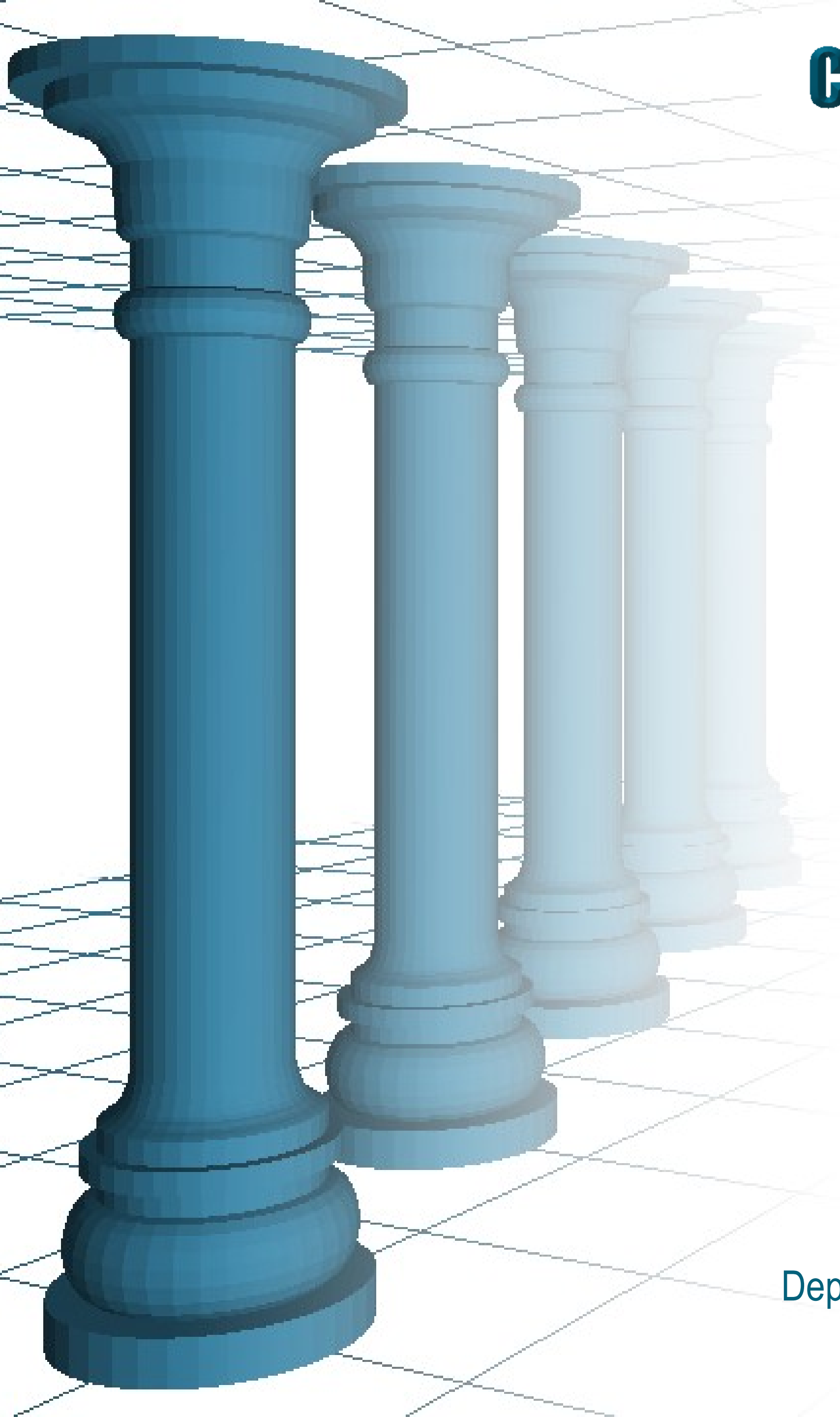


COSC363 Computer Graphics

9

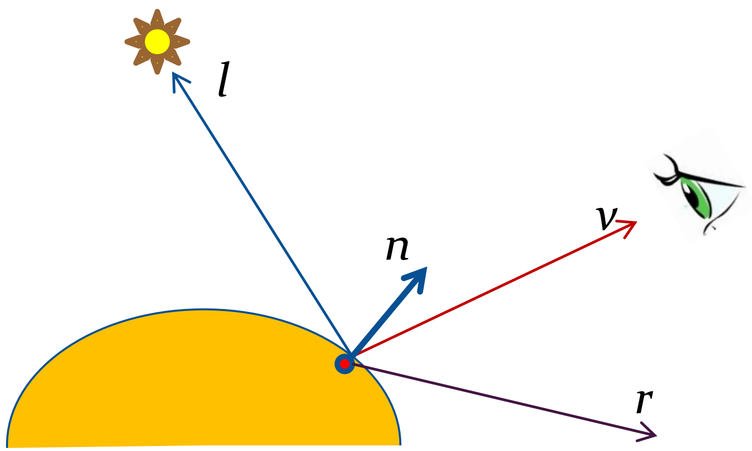
Ray Tracing

A transportation network for light

R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

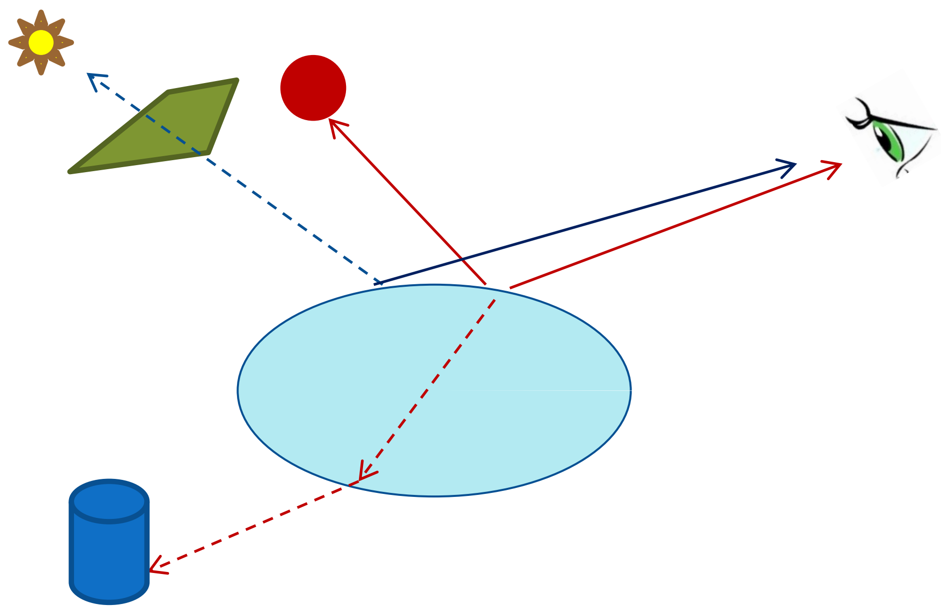
Local Illumination Model

- A local illumination model considers only light travelling from a light source to a surface and then reflected off the surface to the eye.
  - Requires only the light source coordinates, local surface geometry and the material characteristics at a vertex.
  - Suitable for the hardware pipeline (OpenGL, Direct3D)
- Does not consider occlusions and transmittance of light.



Global Illumination

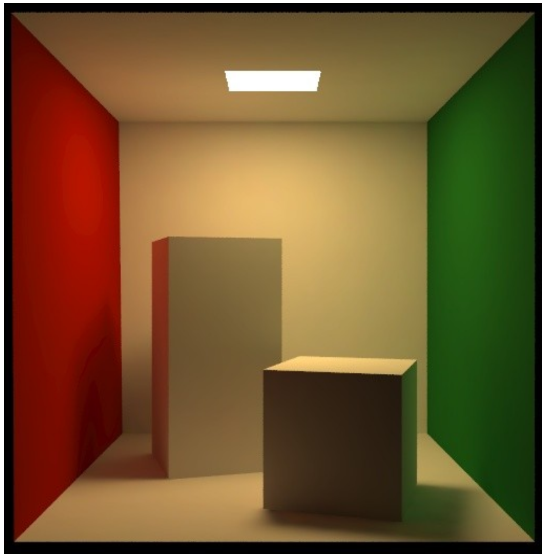
- The illumination at a given point is a combination of the light received from a source and the light reflected from other surfaces in the scene.
- Considers the effects of occlusions, surface reflections, light transmission through a medium (direct transmittance and refractions), and indirect illumination.



Global Illumination Methods

Radiosity:

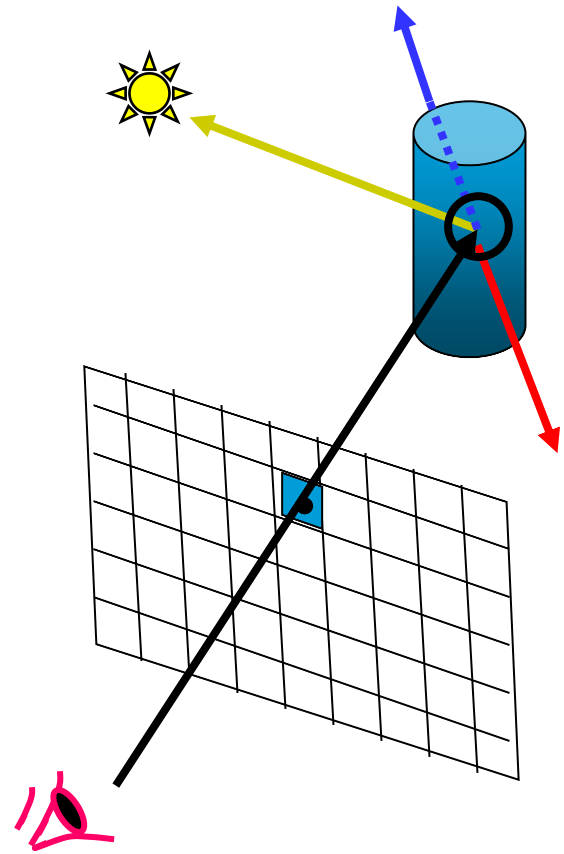
- A scene illumination can be considered as an equilibrium state for radiant energy transfers between surface elements.
- Gives good results for diffuse illumination, but specularity is not handled.
  - Useful for modelling area light sources, colour bleeding, soft shadows.



Cornell Box

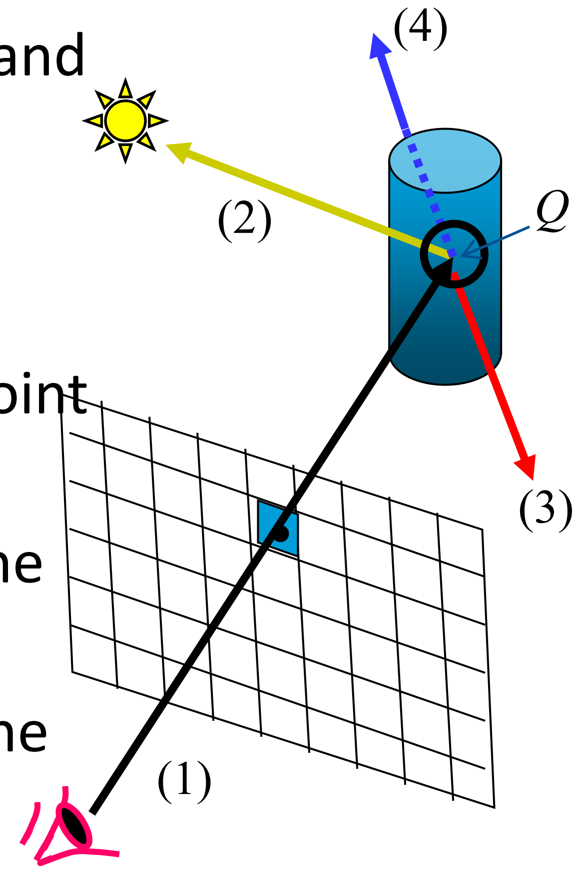
Ray Tracing (Backward Ray Tracing)

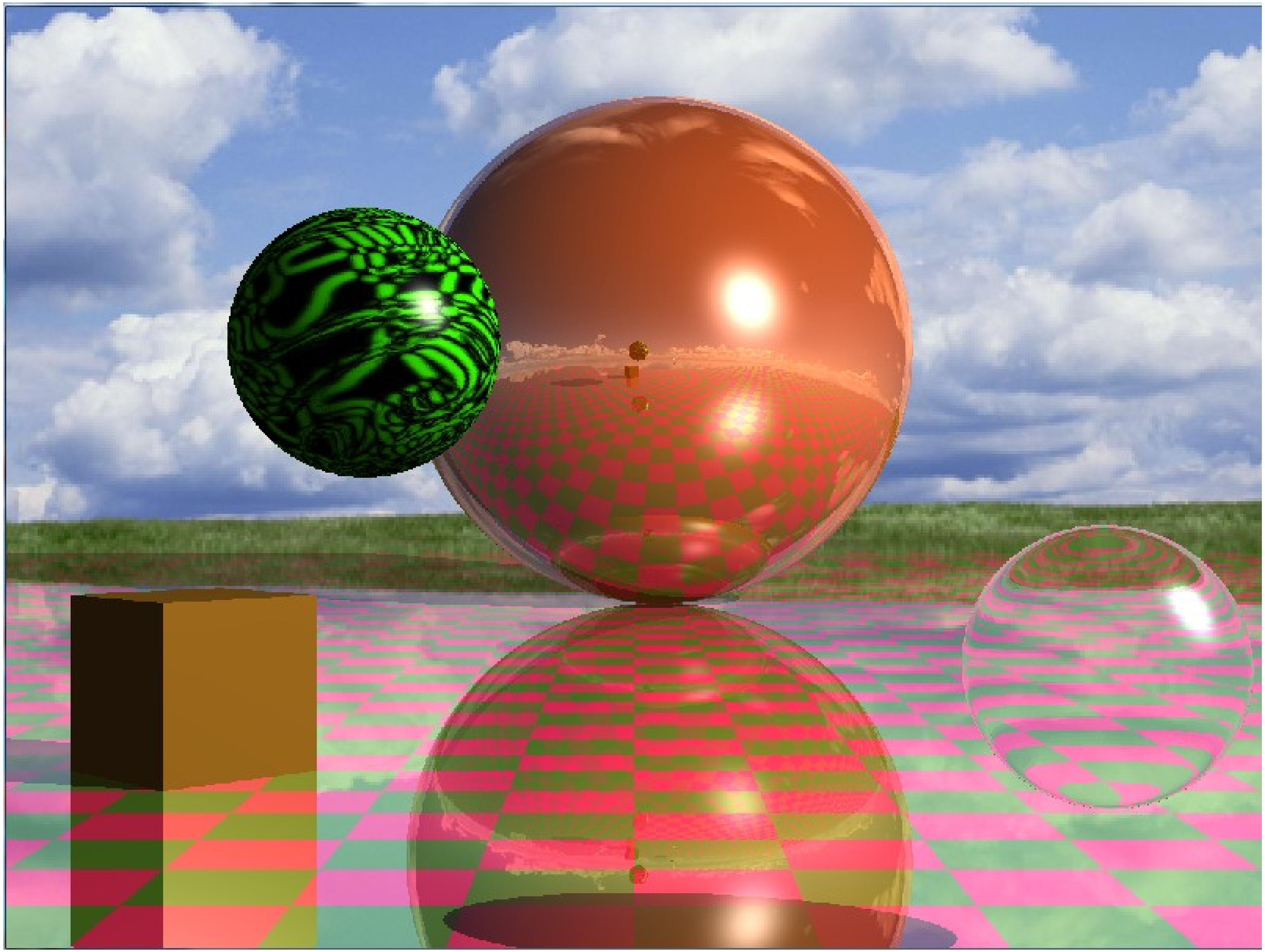
- Traces ray outwards from the eye to objects and light sources. The opposite of reality.
- Use secondary rays to determine shadows and to model mirror reflection and refraction
- We can easily generate many global illumination effects
- But:
  - Doesn't handle diffuse inter-reflections
    - e.g. "colour bleed" from a bright red wall to an adjacent white wall
  - Computationally expensive



“Tracing” a ray

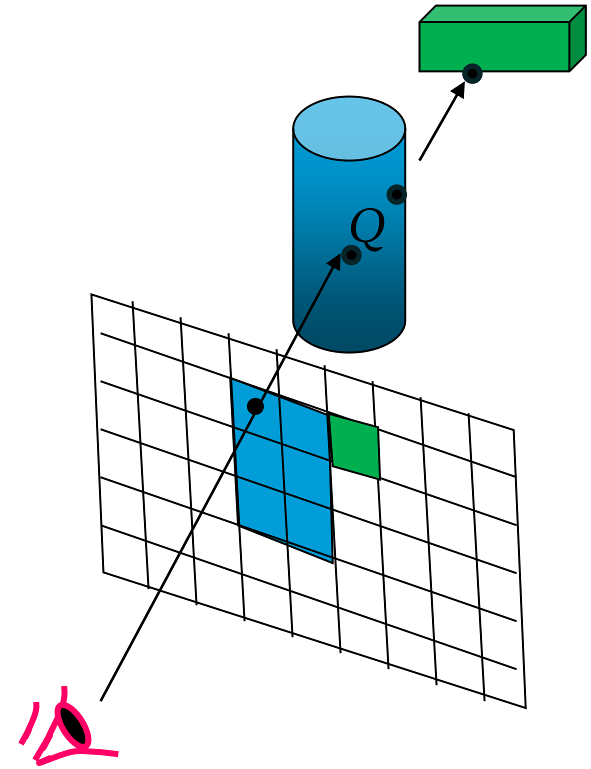
- Compare the ray will all scene objects and compute the closest point of intersection *Q*, and obtain the intersecting object’s index.
- Compute the colour value at the point of intersection *Q*.
- Generate a shadow ray to determine if the point *Q* is in shadow (2)
- If the surface is reflective, recursively trace the reflected ray at *Q* (3)
- If the surface is refractive, recursively trace the refracted ray at *Q* (4)
- Add the colour contributions from (2), (3) and (4), and return the colour value.

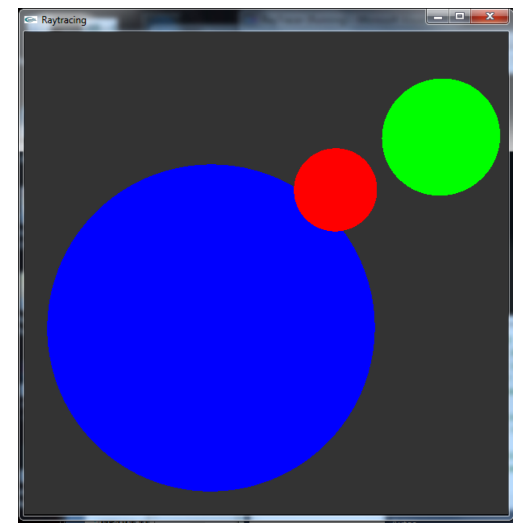




Ray Casting

- Ray tracing without secondary rays.
- Trace a ray from the view point (called the primary ray) through each “pixel” on the image plane
  - Test each surface to determine if it is intersected by the ray.
  - Compute the points of intersection on each primary ray.
  - Get the point of intersection *Q* that is closest to the eye.
  - Use the colour of the object on which *Q* lies as pixel colour.





Ray Casting + Phong Lighting

- At the point of intersection *Q*, compute the colour value using an illumination model.
- Simplified Phong Illumination:

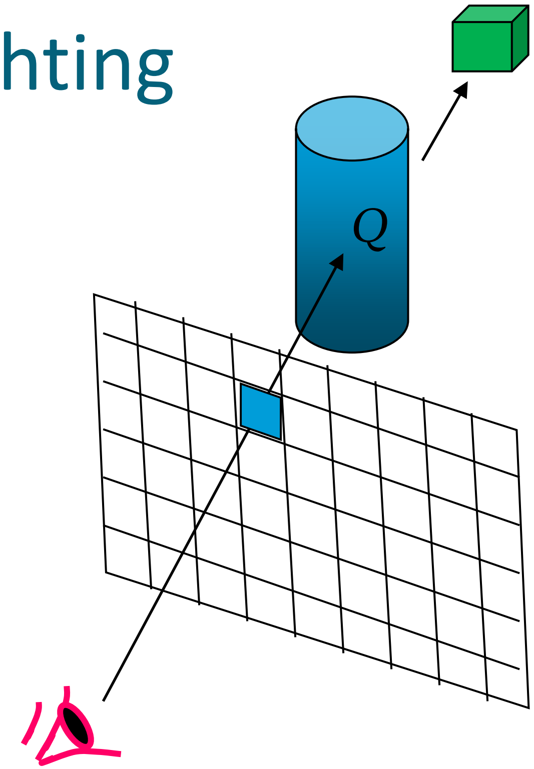
  Assumptions:

  Light's ambient color A = (0.2, 0.2, 0.2)

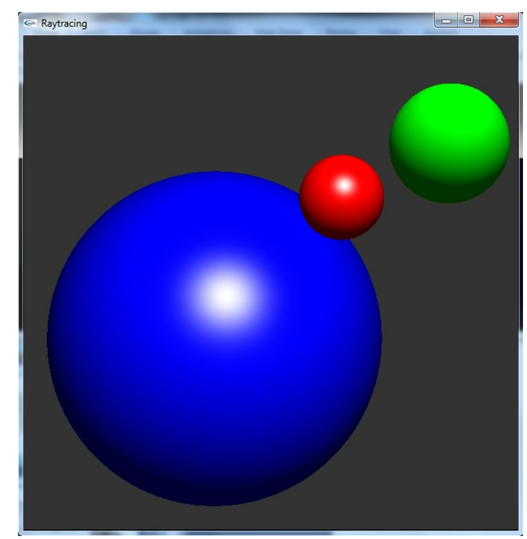
  Light's diffuse and specular color = (1,1,1)

  M = Material Color (ambient and diffuse)

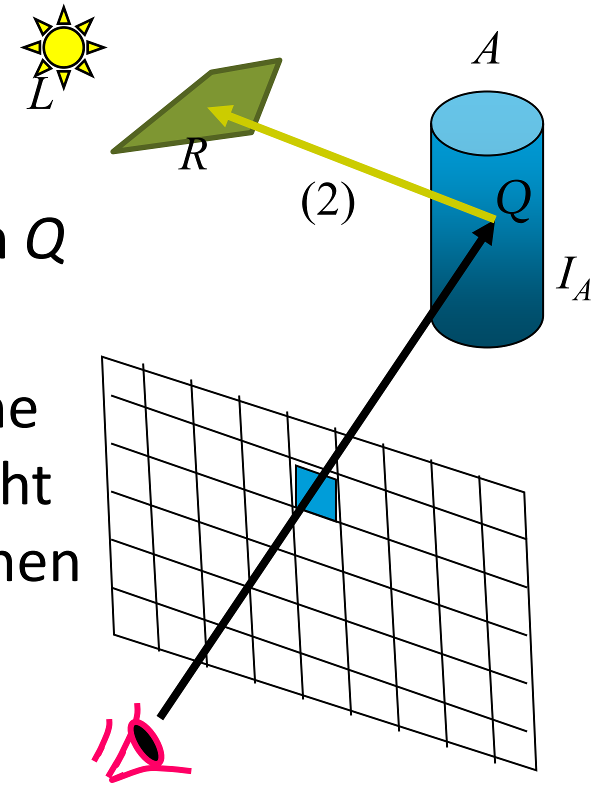
  Material's specular color = (1, 1, 1)



$$\text{Col} = \underbrace{AM}_{\text{Ambient}} + \underbrace{M\,(\boldsymbol{l}\bullet\boldsymbol{n})}_{\text{Diffuse}} + \underbrace{(1, 1, 1)\,(\boldsymbol{r}\bullet\boldsymbol{v})^f}_{\text{Specular}}$$



Shadow Ray



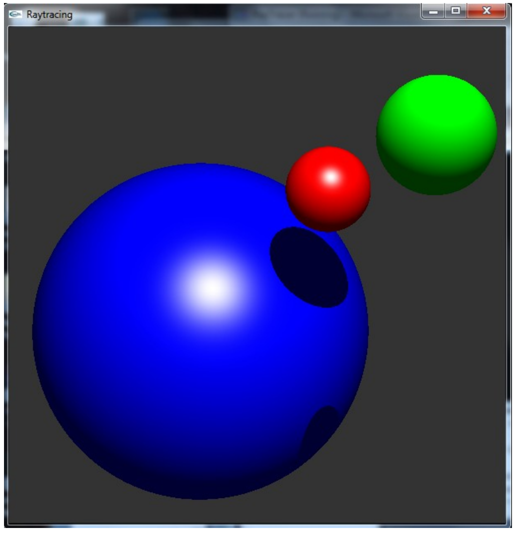
- Trace a ray from the point of intersection *Q* towards the light source *L* (2)
- If the shadow ray hits an object, and if the point of intersection *R* is between the light source and the object *A* (i.e., $RQ < LQ$), then the point *Q* is in shadow.

If *Q* is in shadow

$$I_A = AM$$

Else

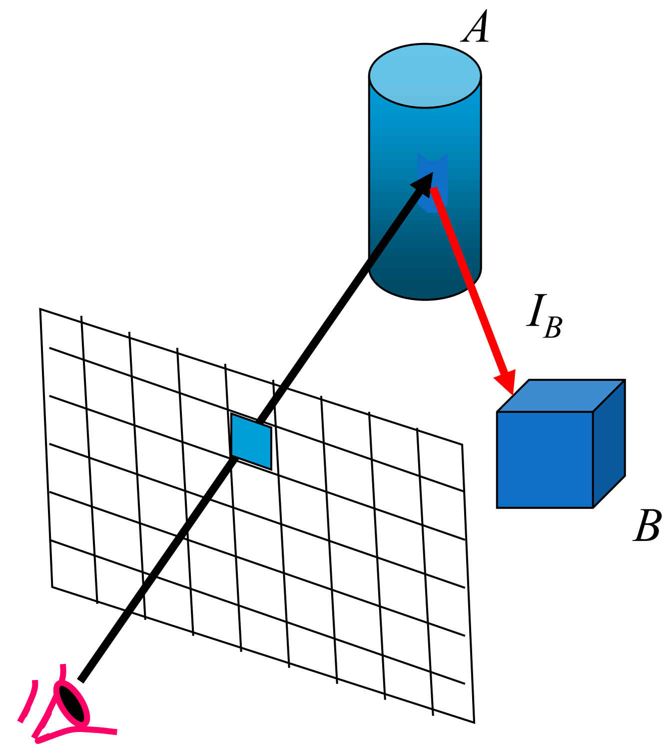
$$I_A = AM + M\,(\boldsymbol{l}\bullet\boldsymbol{n}) + (1, 1, 1)\,(\boldsymbol{r}\bullet\boldsymbol{v})^f$$

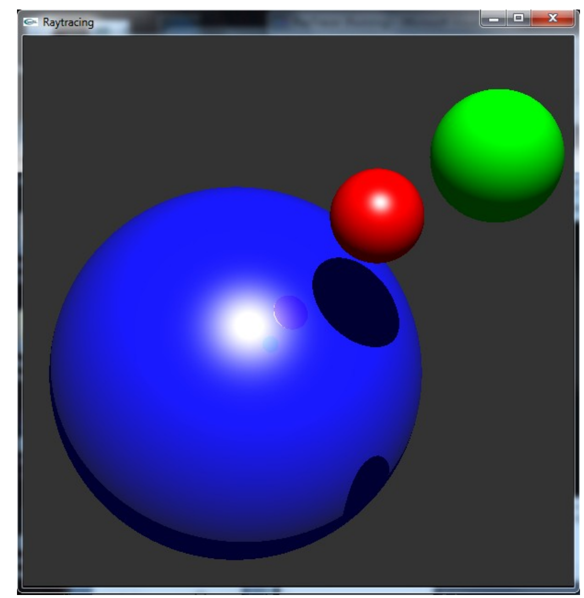


Reflections

- If the surface is reflective, then a secondary ray along the direction of reflection is traced.
- If this secondary ray meets a surface at a point with intensity I_B , then $\rho_r I_B$ is added to the pixel color
  - ρ_r is a scale factor (< 1), called the coefficent of reflection
  - ρ_r represents how much of colour I_B is reflected on the surface A.
- The colour of the pixel is now

$$I = I_A + \rho_r I_B$$



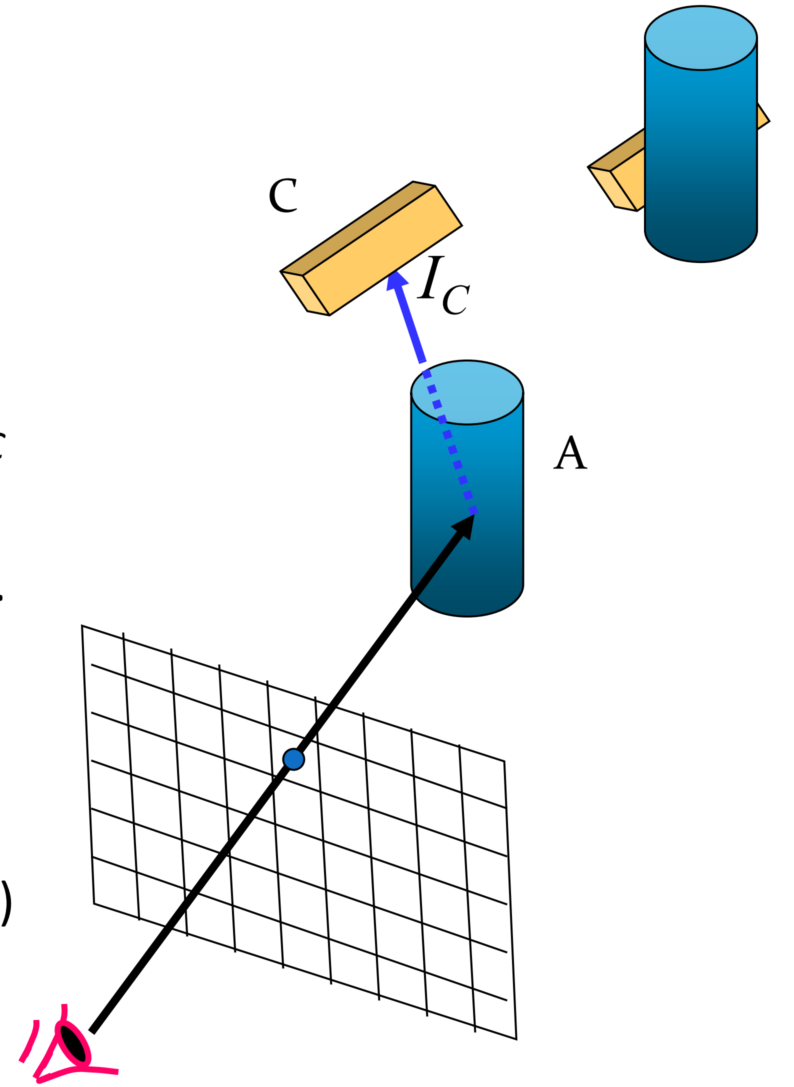


Refractions

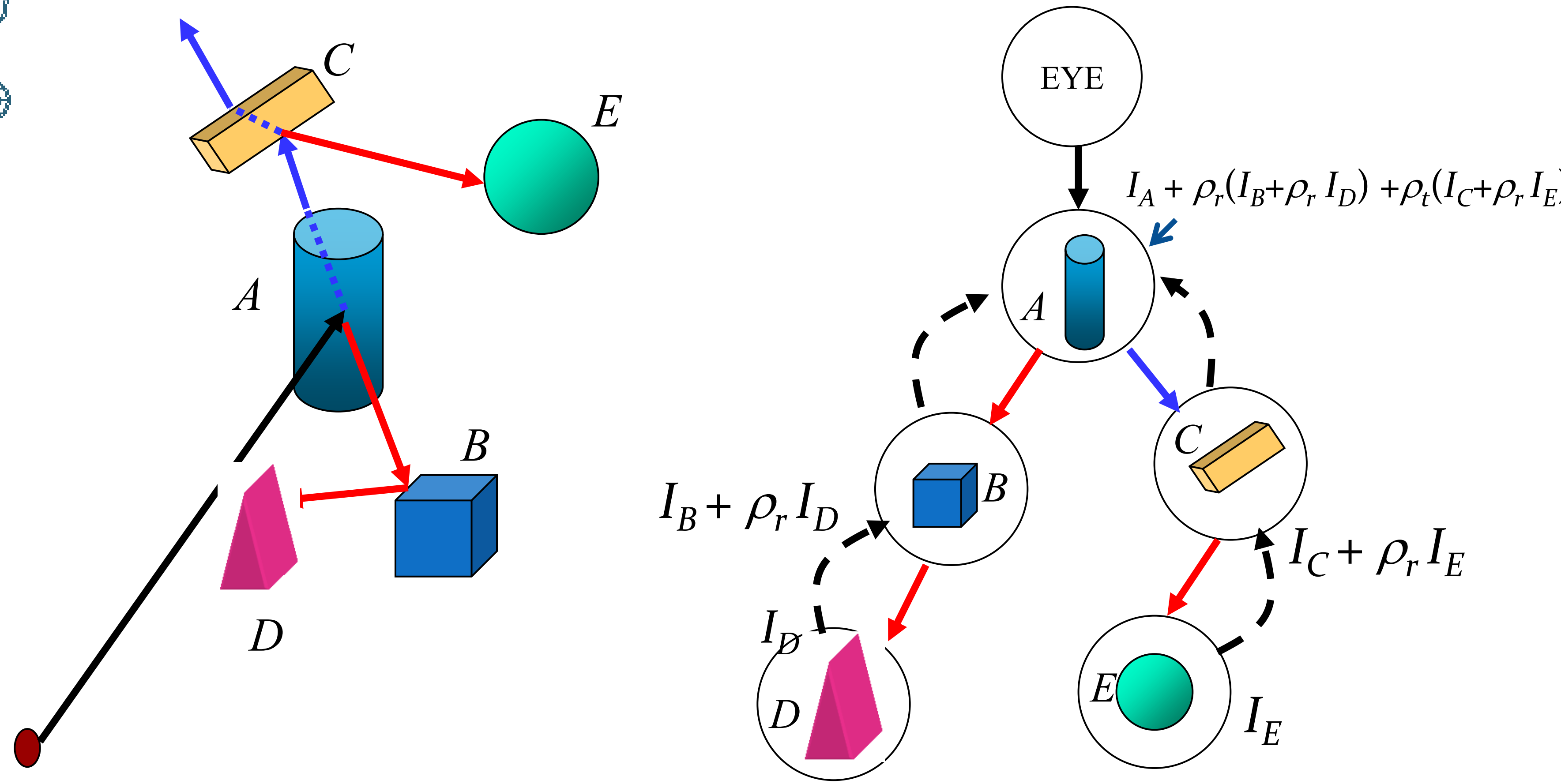
- If the surface is transparent, a secondary ray along the direction of refraction is traced.
- If this secondary ray meets a surface at a point with intensity I_C, then $\rho_t I_C$ is added to the pixel color.
  - ρ_t is a scale factor (<1), called the coeff. of transmission
- The colour of the pixel is now

$$I = I_A + \rho_t I_c$$

(Compare with a similar equation on Slide 12)



Binary Ray-tracing Tree



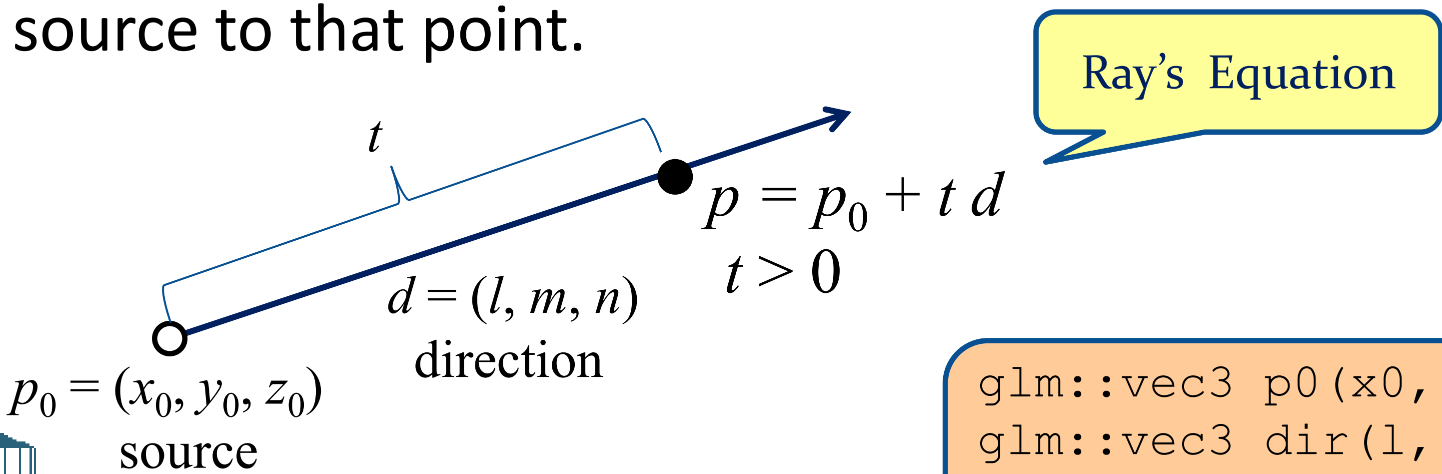
- Left branches represent reflections
- Right branches represent transmission paths

What is a "ray"?

A ray is specified using

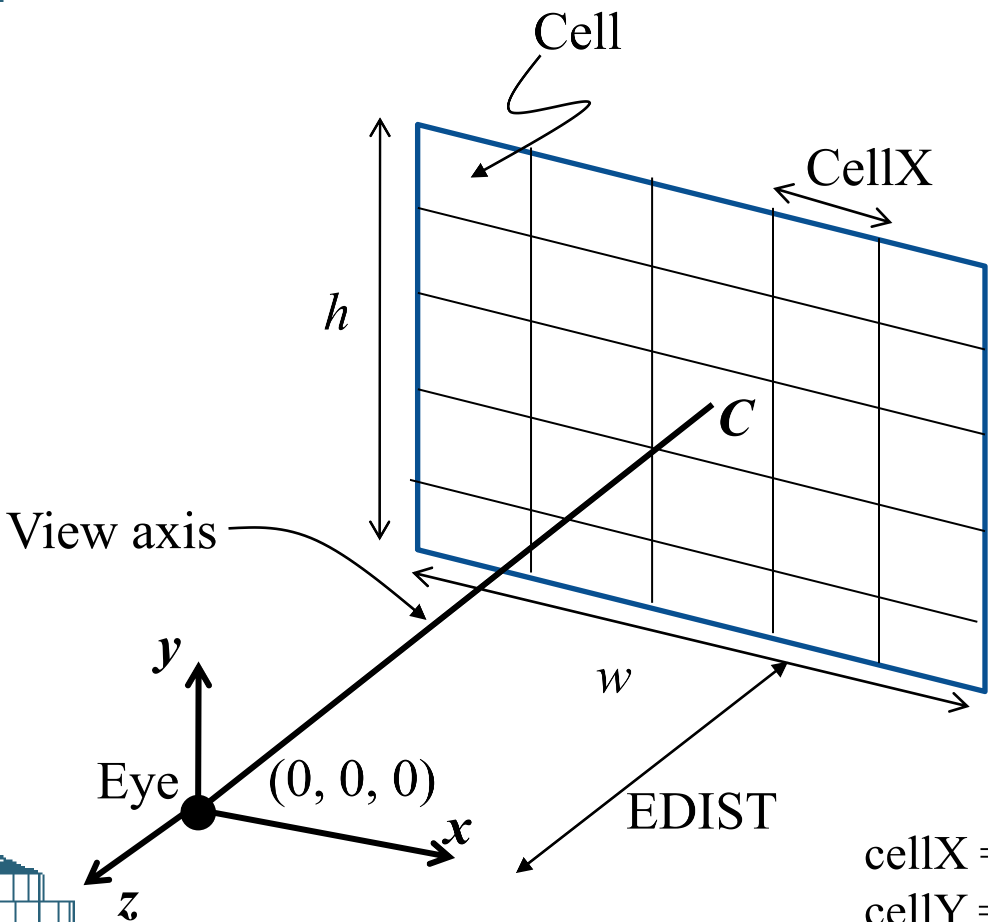
- A point (the source of the ray): $p_0 = (x_0, y_0, z_0)$
- A vector (the direction of the ray): $d = (l, m, n)$
- The vector must be converted to a unit vector.

Any point on the ray can be represented using a single parameter t. The value of t denotes the distance from the source to that point.




```
glm::vec3 p0(x0, y0, z0);
glm::vec3 dir(l, m, n);
Ray ray = Ray(p0, dir);
ray.normalize();
```

Ray Tracing Setup

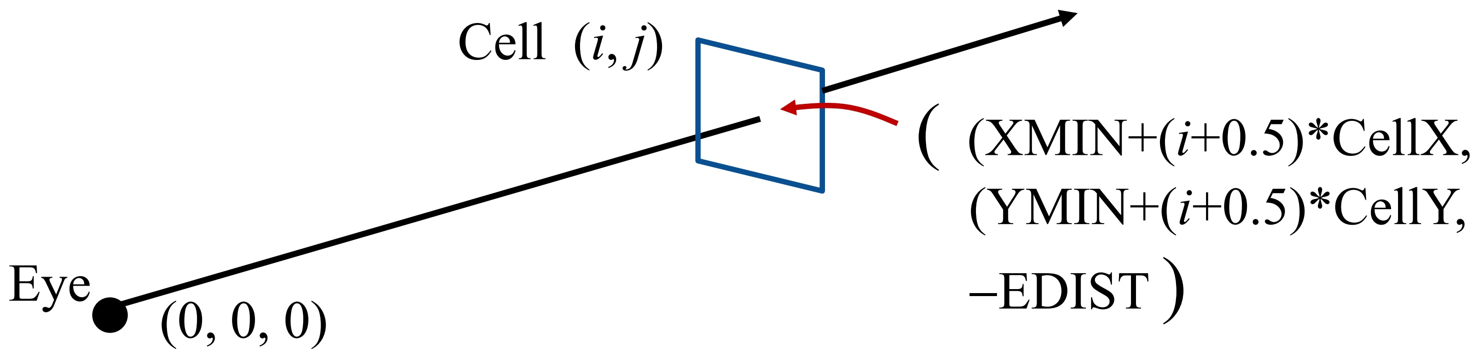


w = Width of screen in world units (eg. 5 units).
h = Height of screen in world units (eg. 5 units).
EDIST = Eye dist in world units (eg. 10 units)
XMIN = $-w/2$; XMAX = $w/2$;
YMIN = $-h/2$; YMAX = $h/2$;

NUMDIV = Number of subdivisions along x, y directions.

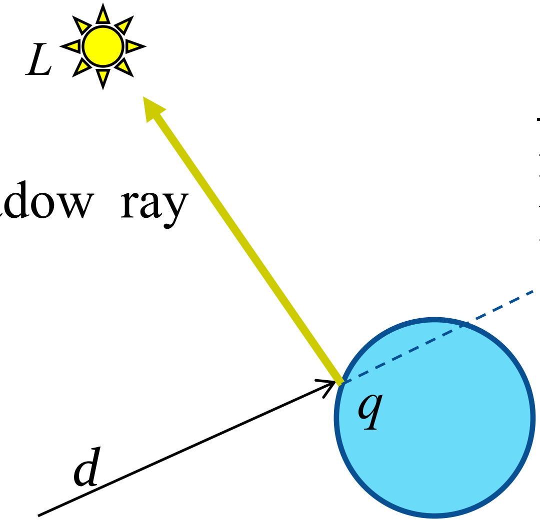
cellX = cell width = (XMAX-XMIN)/NUMDIV
cellY = cell height = (YMAX-YMIN)/NUMDIV
cell indices: i, j = 0, …, NUMDIV-1.

Primary Ray



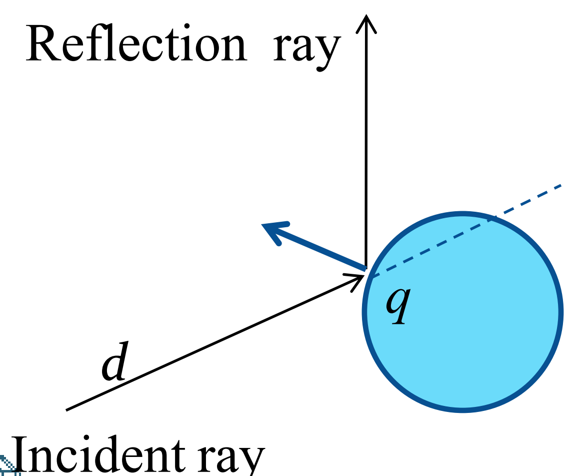
- Ray position: (0, 0, 0)
- Ray direction *d*: ((XMIN+(i+0.5)*CellX, (YMIN+(i+0.5)*CellY, –EDIST)
- Normalize the above direction.

Secondary Rays



<u>Shadow ray</u>: A ray that originates at a point of intersection with an object, and directed towards a light source.

Position = q (point of intersection)
Direction = $L - q$ normalized.



<u>Reflection ray</u>: A ray from the intersection point towards the direction of reflection of the incident ray (Used only for reflective surfaces)

Position = q
Direction:
$r = -2(d \bullet n)n + d$ normalized.
(We will use a GLM function to compute r)

GLM Functions for Ray Tracing

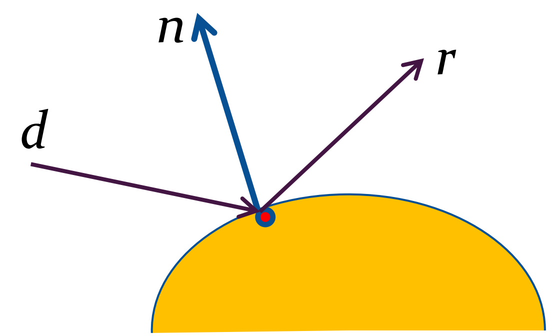
- Reflection:

  r = glm::**reflect**(***d***, ***n***);

  d: Unit incident vector

  n: Unit normal vector

  The reflection vector also is a unit vector.



- Refraction:

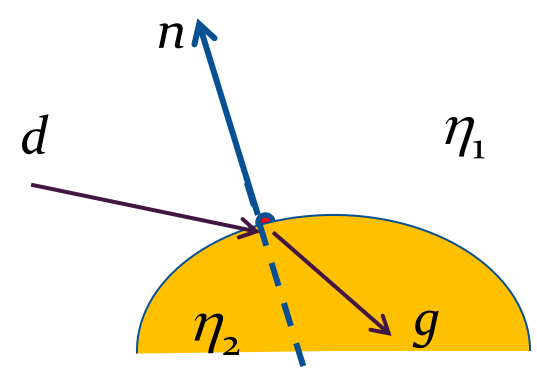
  g = glm::**refract**(***d***, ***n*,** eta);

  d: Unit incident vector

  n: Unit normal vector

  The refraction vector also is a unit vector.

  eta = Ratio of refractive indices = η_1/η_2



Reflections

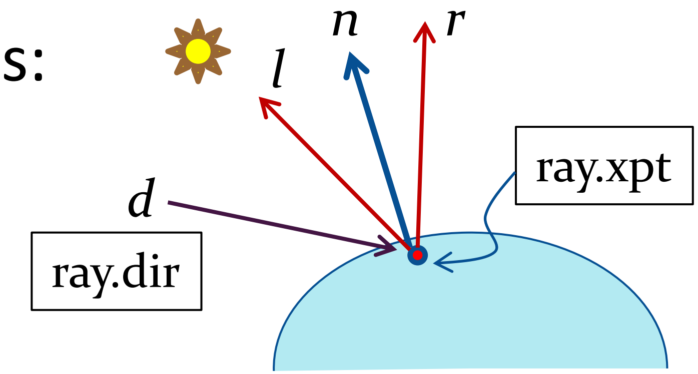
- Computation of specular reflections:

  r = glm::**reflect**(−***l***, ***n***);

  l: <u>Unit</u> light source vector

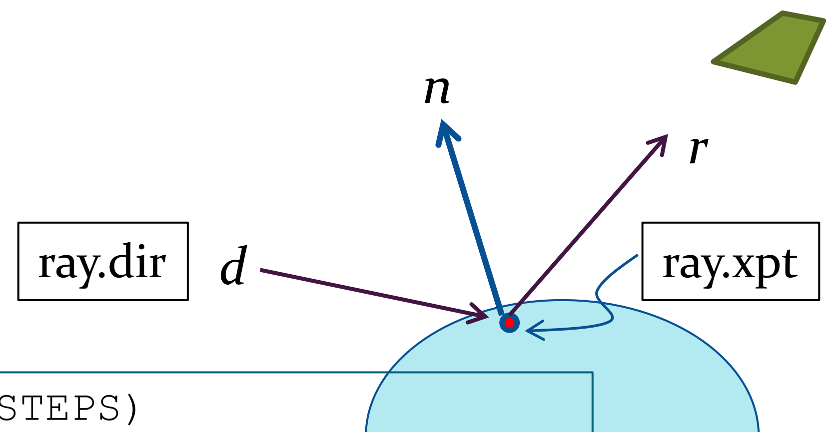
  Specular term = $(\boldsymbol{r}.\boldsymbol{v})^f$.

  v: View vector = −***d***



- Surface reflections:

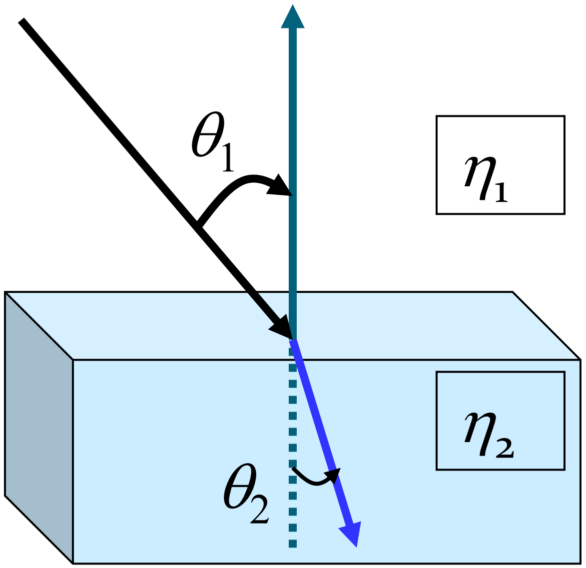
  r = glm::**reflect**(***d***, ***n***);




```
if(ray.xindex == 0 && step < MAX_STEPS)
{
  glm::vec3 refl = glm::reflect(ray.dir, normalVector);
  Ray reflectedRay(ray.xpt, refl);
  glm::vec3 reflCol = trace(reflectedRay, step+1);
  colorSum = colorSum + (0.8f*reflCol);
}
```

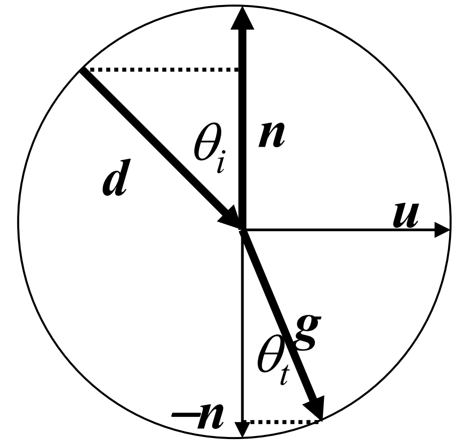
Index of Refraction

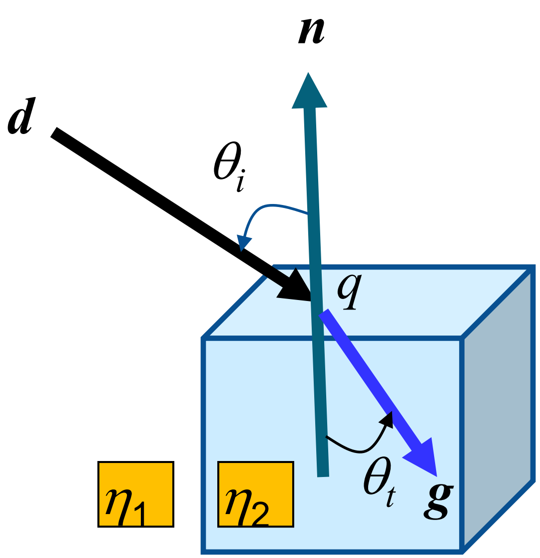
- Light travels at speed c/η in a medium with index of refraction η.
- Common values of index of refraction:
  - Air 1.
  - Water 1.33
  - Glass 1.5
  - Diamond 2.4



- Snell's Law of Refraction:
  - $\eta_1 \sin\theta_1 = \eta_2 \sin\theta_2$

Refracted Ray



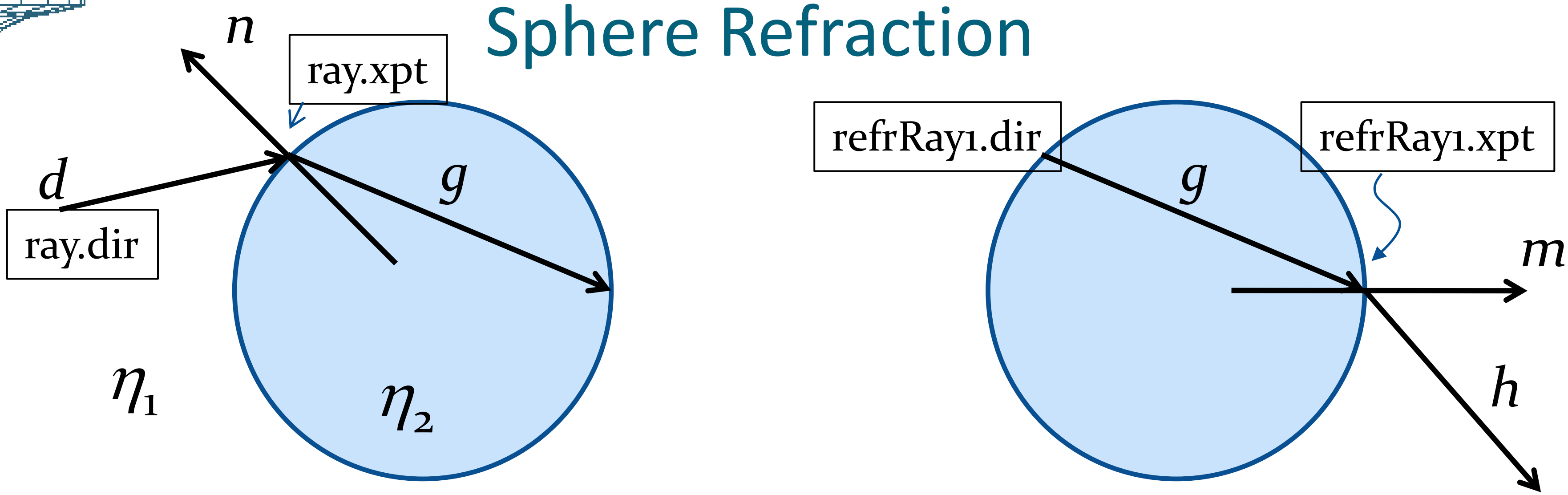


$$\boxed{\begin{aligned}\boldsymbol{u}\sin\theta_i - \boldsymbol{n}\cos\theta_i &= \boldsymbol{d}\\ \boldsymbol{u}\sin\theta_t - \boldsymbol{n}\cos\theta_t &= \boldsymbol{g}\end{aligned}}$$

$$\boldsymbol{g} = \left(\frac{\eta_1}{\eta_2}\right)\boldsymbol{d} - \left(\frac{\eta_1}{\eta_2}(\boldsymbol{d}.\boldsymbol{n}) + \cos\theta_t\right)\boldsymbol{n}$$

$$\cos\theta_t = \sqrt{\left(1 - \left(\frac{\eta_1}{\eta_2}\right)^2\left(1 - (\boldsymbol{d}.\boldsymbol{n})^2\right)\right)}$$

Sphere Refraction



n = sceneObjects[ray.xindex]->normal(ray.xpt);

g = glm::**refract**(***d***, ***n*,** eta);

Ray refrRay1(ray.xpt, ***g***)

refrRay1.closestPt(sceneObjects);

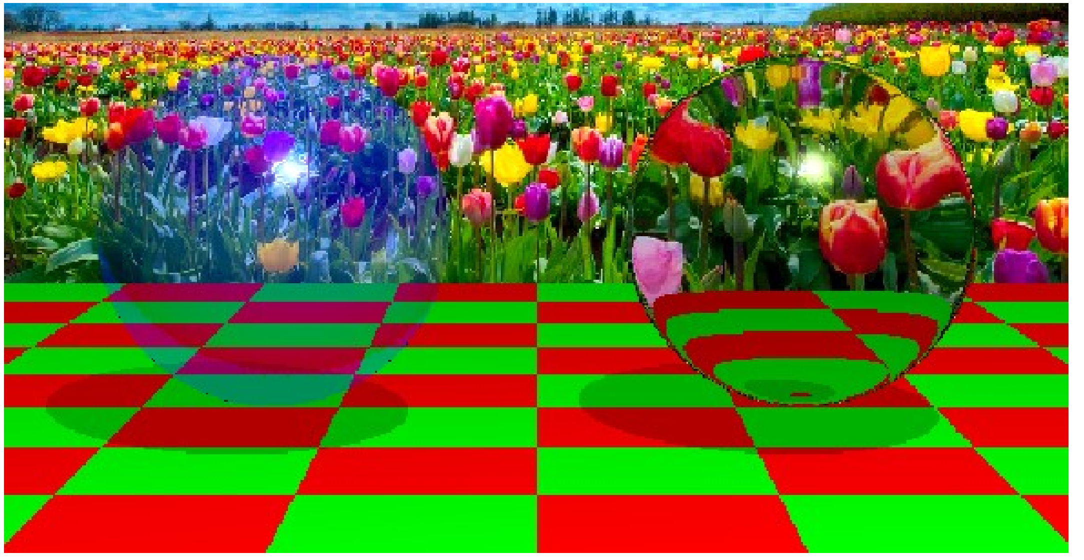
m = sceneObjects[refrRay1.xindex]->normal(refrRay1.xpt);

h = glm::**refract**(***g***, ***–m*,** 1.0f/eta);

Sphere Refraction

eta = 1/1.5

Sphere Refractions



eta = 1/1.01

eta = 1/1.003

Shadows

Bad

Good

Better

Ray-plane intersection

- Given a point a on a plane, and the normal vector n of the plane, the plane's equation can be written as

  $(p - a) \bullet n = 0$

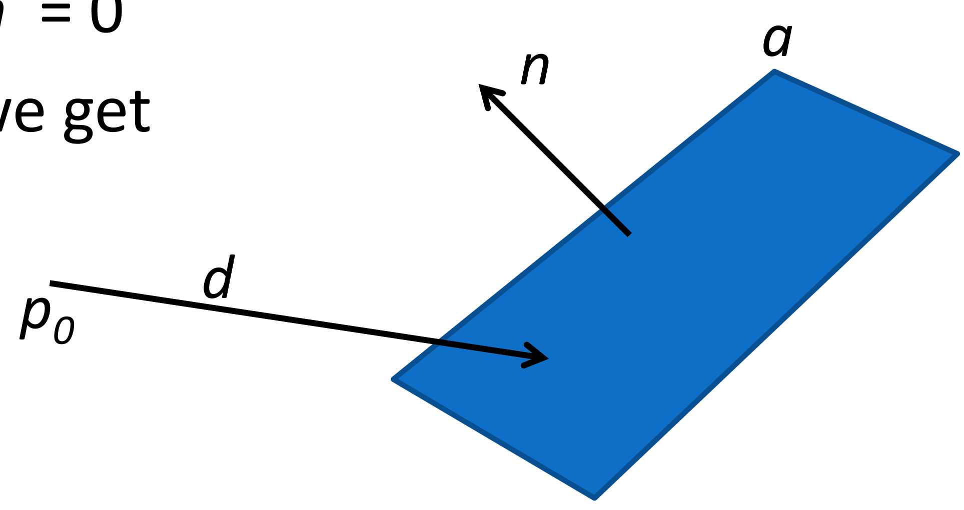
- A ray is given by the equation

  $p = p_0 + t\,d.$

- At the point of intersection, both equations are true. Therefore, $(p_0 + t\,d - a) \bullet n = 0$

- From the above equation, we get

$$t = \frac{(a - p_0) \bullet n}{d \bullet n}$$



Ray-sphere intersection

- Equation of a sphere centred at C with radius r is

  $(p - C) \bullet (p - C) = r^2$
- Consider a ray given by the equation

  $p = p_0 + t\,d.$
- At the point of intersection, both equations are true.

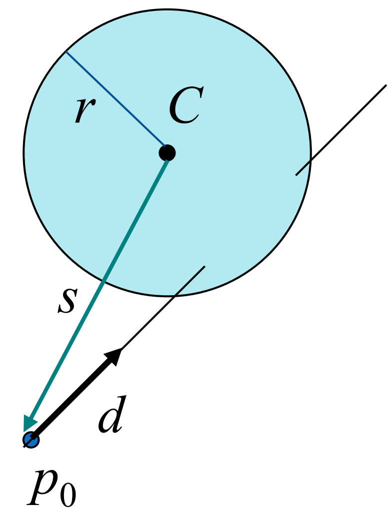
Therefore, $(p_0 + t\,d - C) \bullet (p_0 + t\,d - C) = r^2$

$(s + t\,d) \bullet (s + t\,d) = r^2$, where $s = p_0 - C$

$(d \bullet d)\, t^2 + 2\,(s \bullet d)\, t + (s \bullet s) - r^2 = 0.$

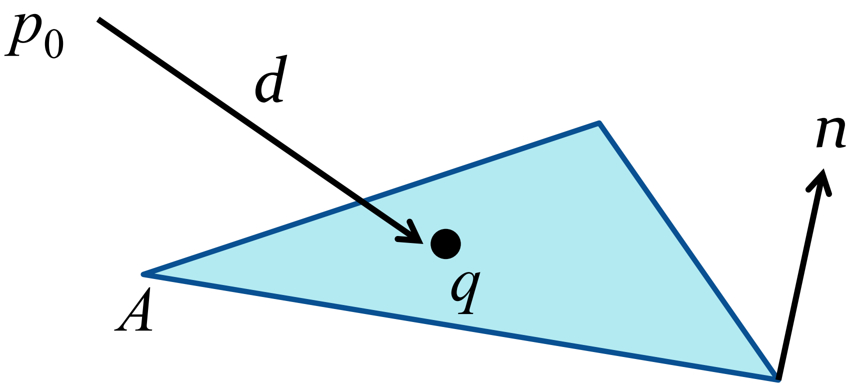
Since d is a unit vector, we get

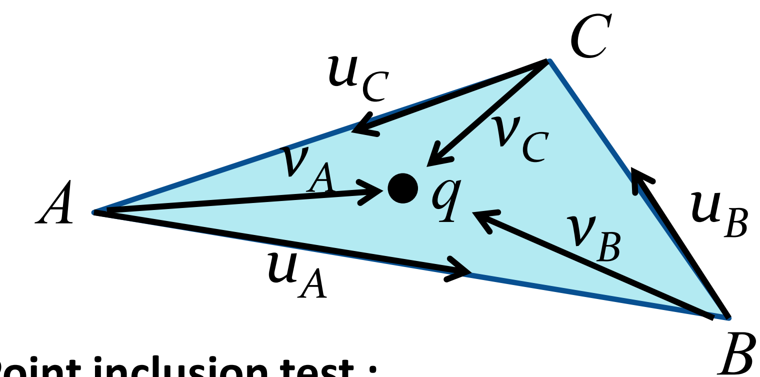
$$t = -(s \bullet d) \pm \sqrt{(s \bullet d)^2 - (s \bullet s) + r^2}$$



Ray intersection with polygonal objects

- Use the plane equation of each triangle to get the point of intersection with the ray.
- Check if the point of intersection lies within the triangle.





Intersection :

$$t = \frac{(A - p_0)\cdot n}{d \cdot n}$$

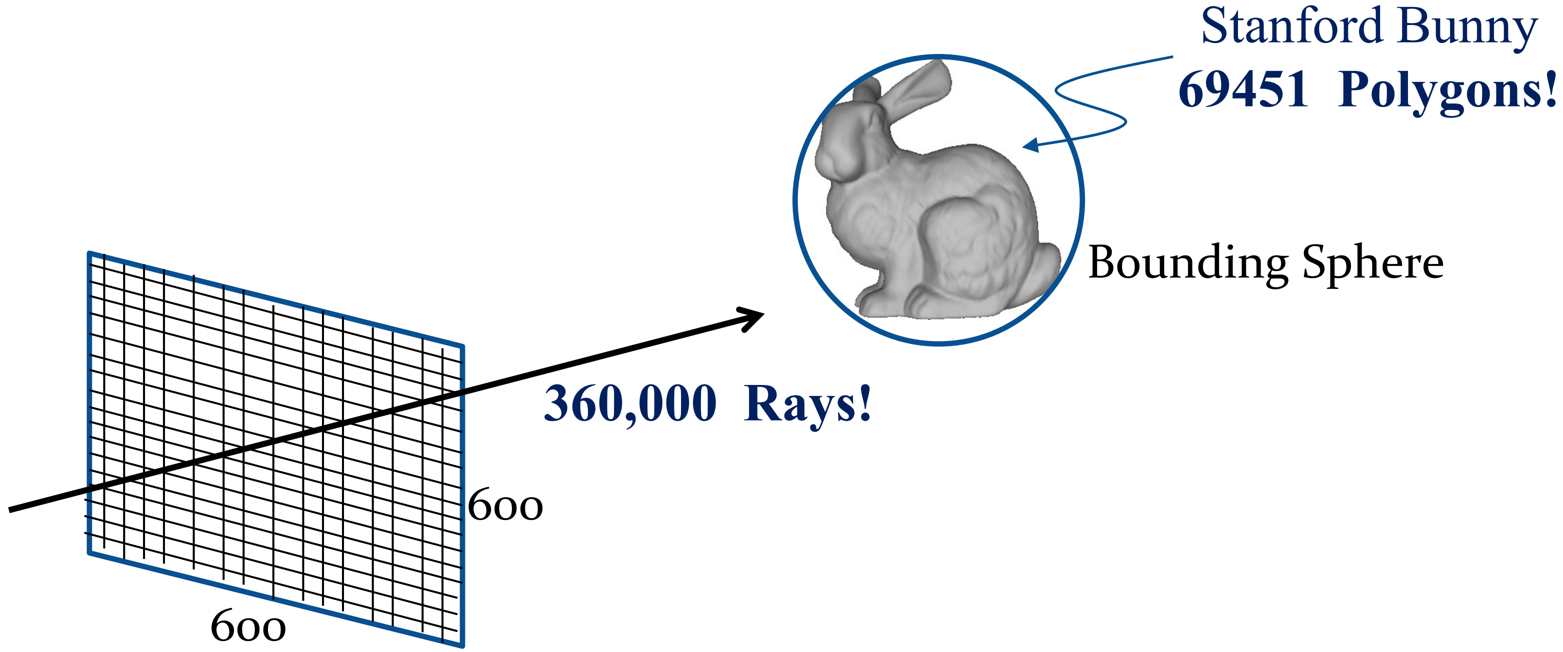
$$q = p_0 + \; t\, d$$

Point inclusion test :

If the cross products
$(u_A \times v_A)$, $(u_B \times v_B)$, $(u_C \times v_C)$
have the same sign, then the point q is inside the triangle.

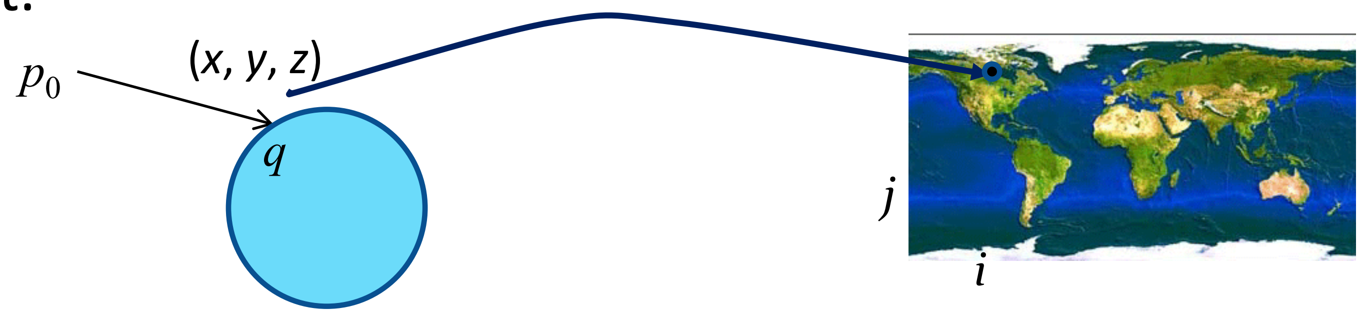
Ray intersection with polygonal objects

- Complex polygonal objects will require a large amount of ray-triangle intersection tests.
- Bounding volume hierarchies and spatial subdivision methods (*k*d-Trees, Octrees) are used to reduce the number of ray-primitive intersection tests.



Texture Mapping

- 2D texturing:
  - Similar to OpenGL texturing, but does not use texture coordinates or texture memory.
  - Map the coordinates of the point of intersection (*x*, *y*, *z*) to image coordinates, and assign the colour of the pixel to that point.



- Procedural texturing:
  - Define functions to map (*x*, *y*, *z*) coordinates to (*r*, *g*, *b*) colour values. $r = r(x, y, z)$; $g = g(x, y, z)$; $b = b(x, y, z)$;

Texture Mapping

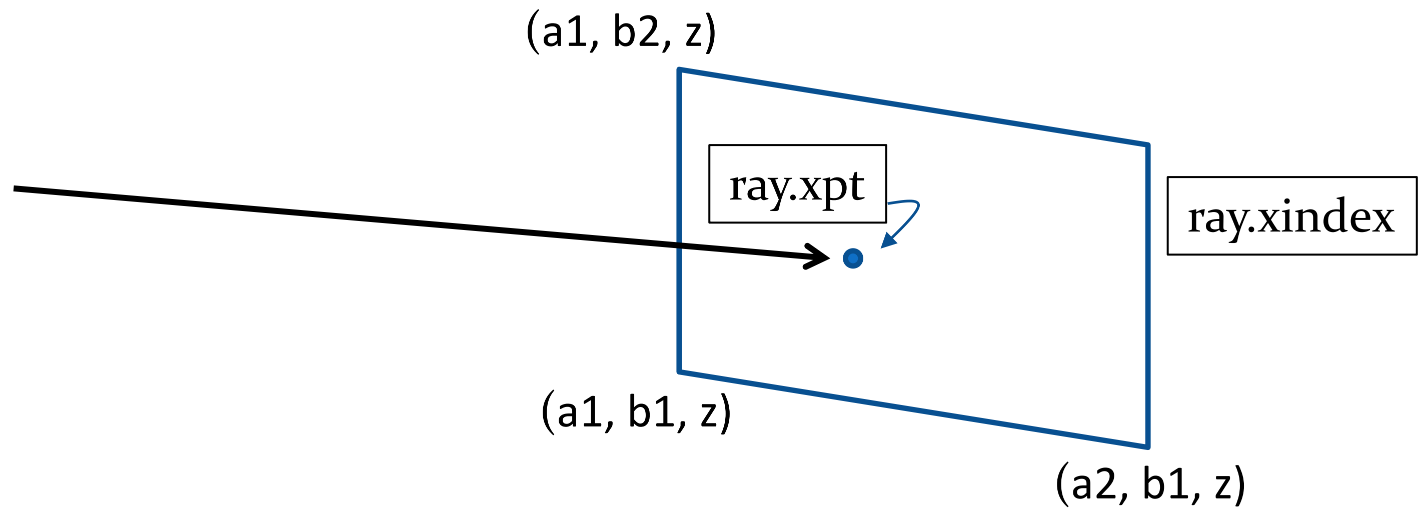
```
...
#include "Ray.h"
#include "TextureBMP.h"
#include <GL/glut.h>
using namespace std;
...
TextureBMP texture;
...
void initialize()
{
   ...
   texture = TextureBMP("myPicture.bmp");
}
...
glm::vec3 trace(Ray ray, int step)
{
   ...
   col = texture.getColorAt(s, t);
}
```

Lab08

TextureBMP.h
TextureBMP.cpp

Note:
$0 \leq s \leq 1$
$0 \leq t \leq 1$

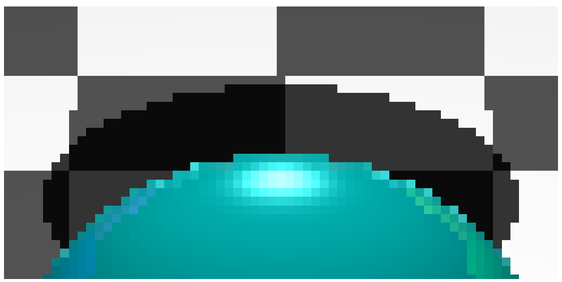
Texturing a Plane

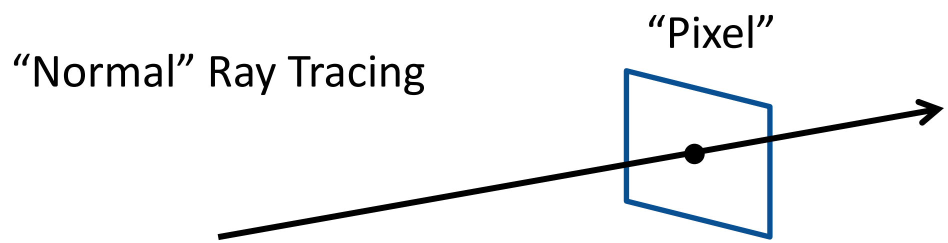



```
if(ray.xindex == 3)
{
   texcoords = (ray.xpt.x - a1)/(a2-a1);
   texcoordt = (ray.xpt.y - b1)/(b2-b1);
   col = texture.getColorAt(texcoords, texcoordt);
}
```

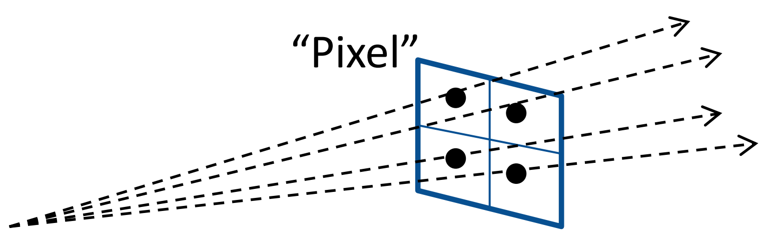
Anti-Aliasing

The ray tracing algorithm samples the light field using a finite set of rays generated through a discretized image space. This results in distortion artefacts such as jaggedness along edges of polygons and shadows.



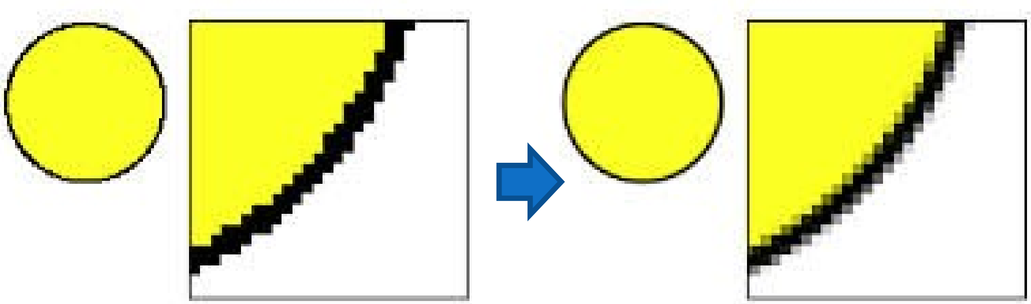


- Supersampling: Generate several rays through each square pixel (eg. divide the pixel into four equal segments) and compute the average of the colour values.



Anti-Aliasing

- Adaptive Sampling: As shown on the previous slide, each pixel is divided into four “sub-pixels”. Primary rays are generated through the centres of each sub-pixel. If the colour value along any ray varies significantly from the other three, that sub-pixel is split further into four sub-pixels, and more rays are generated through them.

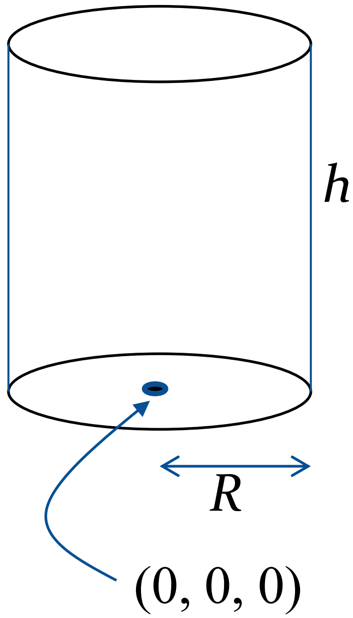


Cylinder

- A cylinder at the origin with axis along the *y*-axis, radius *R* and height *h* is given by

  $x^2 + z^2 = R^2$

  $0 \le y \le h$



- Normal vector at (x, y, z)

  (un-normalized) $n = (x, 0, z)$

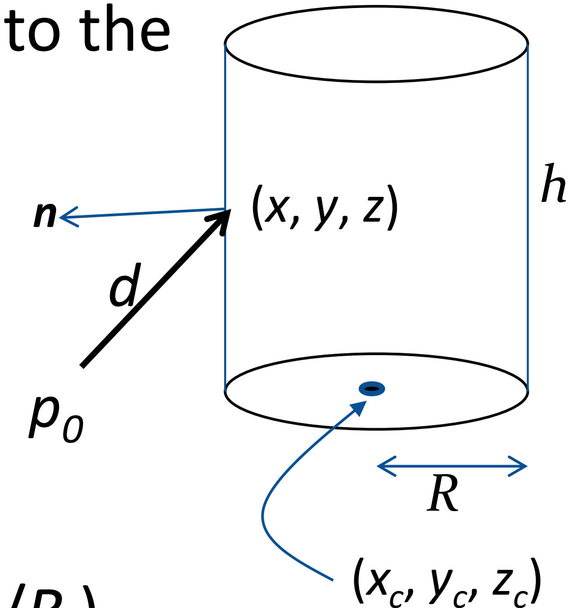
  Normalized $n = (x/R,\ 0,\ z/R)$

Ray – Cylinder Intersection

- A cylinder at (x_c, y_c, z_c), with axis parallel to the y-axis, radius R and height h is given by

$$(x - x_c)^2 + (z - z_c)^2 = R^2$$

$$0 \le (y - y_c) \le h$$



- Normal vector at (x, y, z)

(un-normalized) $\boldsymbol{n} = (x - x_c,\ 0,\ z - z_c)$

(normalized) $\boldsymbol{n} = (\ (x - x_c)/R,\ \ 0,\ \ (z - z_c)/R\ )$

Ray equation:

$x = x_0 + d_x\, t;\qquad y = y_0 + d_y\, t;\qquad z = z_0 + d_z\, t;$

- Intersection equation:

$$t^2\left(d_x^2 + d_z^2\right) + 2t\left\{d_x\left(x_0 - x_c\right) + d_z\left(z_0 - z_c\right)\right\} + \left\{\left(x_0 - x_c\right)^2 + \left(z_0 - z_c\right)^2 - R^2\right\} = 0.$$

Cone

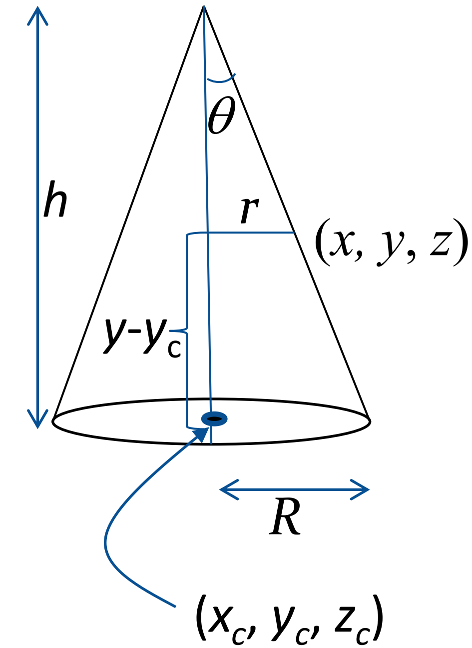
- Consider a cone with centre of base at (x_c, y_c, z_c), axis parallel to the y-axis, radius R, and height h:
- Important equations:

  $\tan(\theta) = R/h$ (θ = half cone angle)

  For any point (x, y, z) on the cone,

  $(x - x_c)^2 + (z - z_c)^2 = r^2$

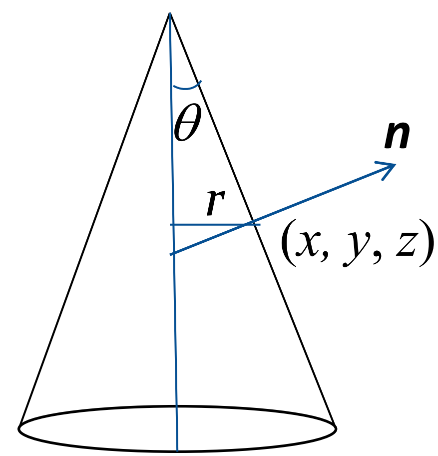
  where, $r = \left(\dfrac{R}{h}\right)(h - y + y_c)$



- Surface normal vector (normalized):

  $\boldsymbol{n} = (\sin\alpha\cos\theta,\ \sin\theta,\ \cos\alpha\cos\theta)$

  where $\alpha = \tan^{-1}\left(\dfrac{x - x_c}{z - z_c}\right)$



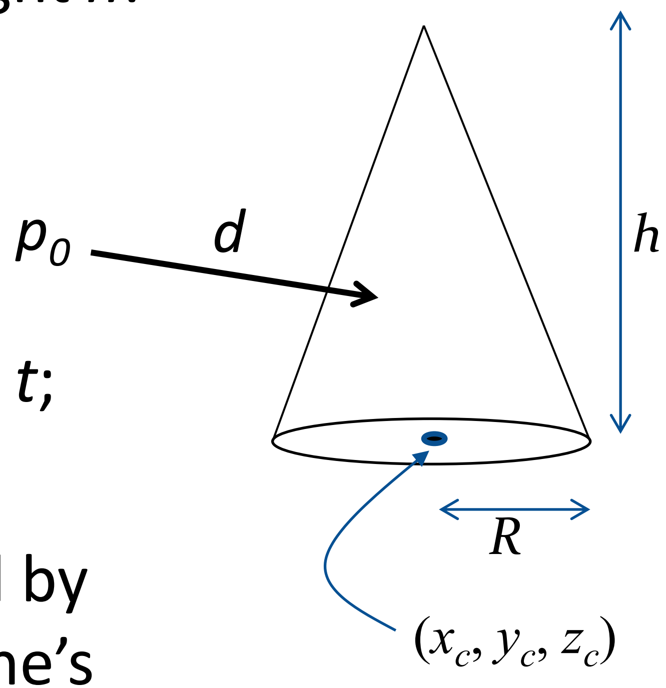
Ray-Cone Intersection

- Equation of a cone with base at (x_c, y_c, z_c), axis parallel to the y-axis, radius R, and height h:

$$(x-x_c)^2+(z-z_c)^2=\left(\frac{R}{h}\right)^2(h-y+y_c)^2$$

- Ray equation:

$x = x_0 + d_x\, t; \quad y = y_0 + d_y\, t; \quad z = z_0 + d_z\, t;$



- The points of intersection are obtained by substituting the ray equation in the cone's equation and solving for t.