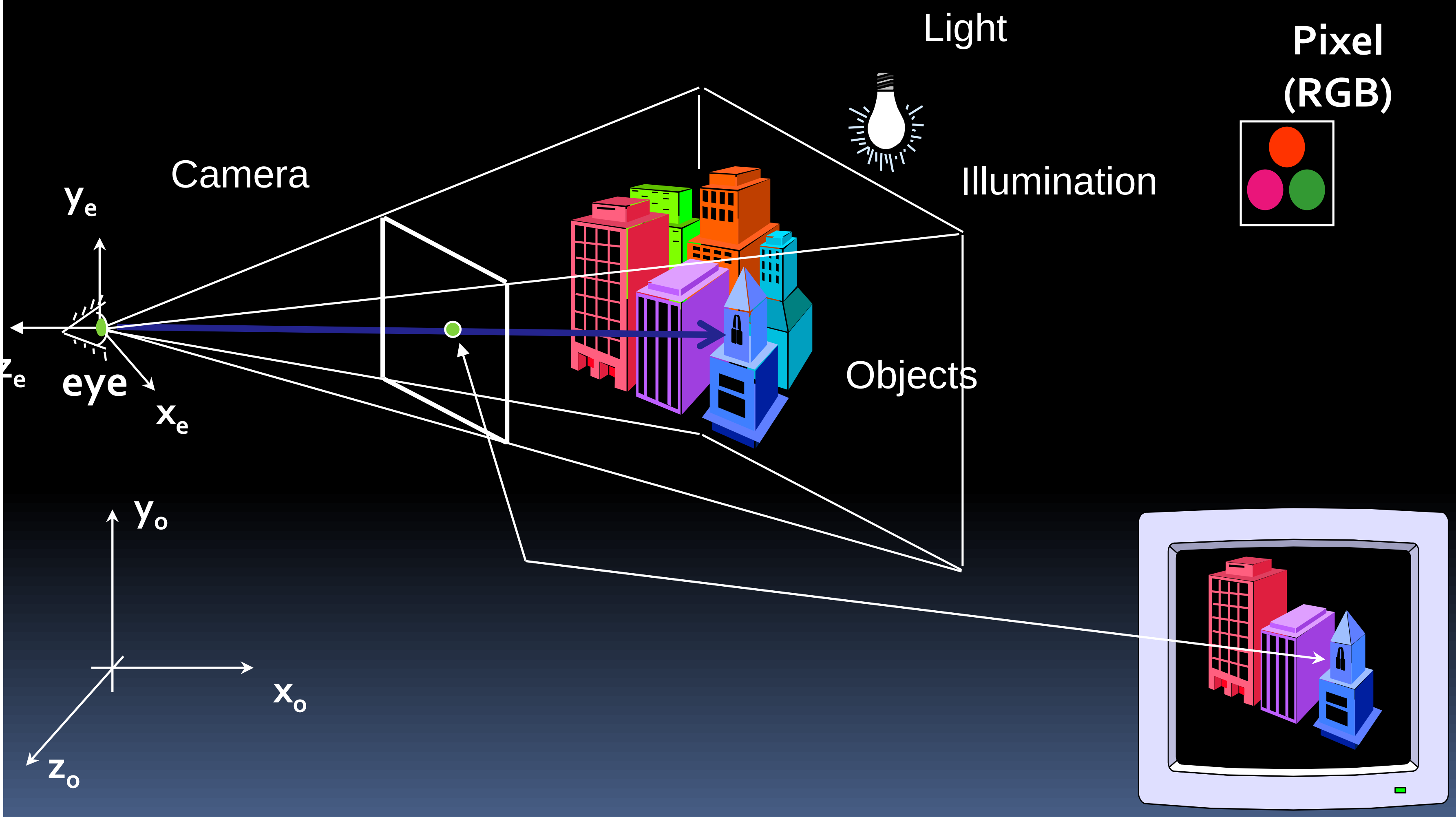
TURNER WHITTED'S RAY-TRACING ALGORITHM: PRACTICE

PROGRAMAÇÃO 3D

MEIC/IST

3D Rendering

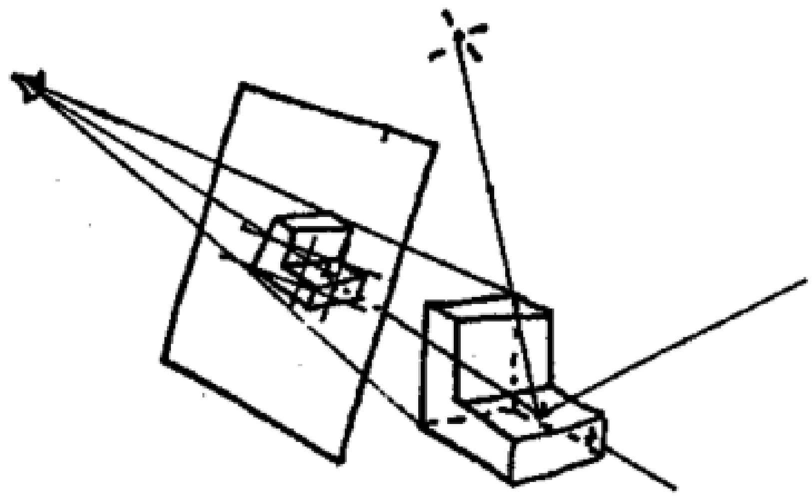


Ray Tracing History

Ray Tracing in Computer Graphics

Appel 1968 - Ray casting

1. Generate an image by sending one ray per pixel
2. Check for shadows by sending a ray to the light



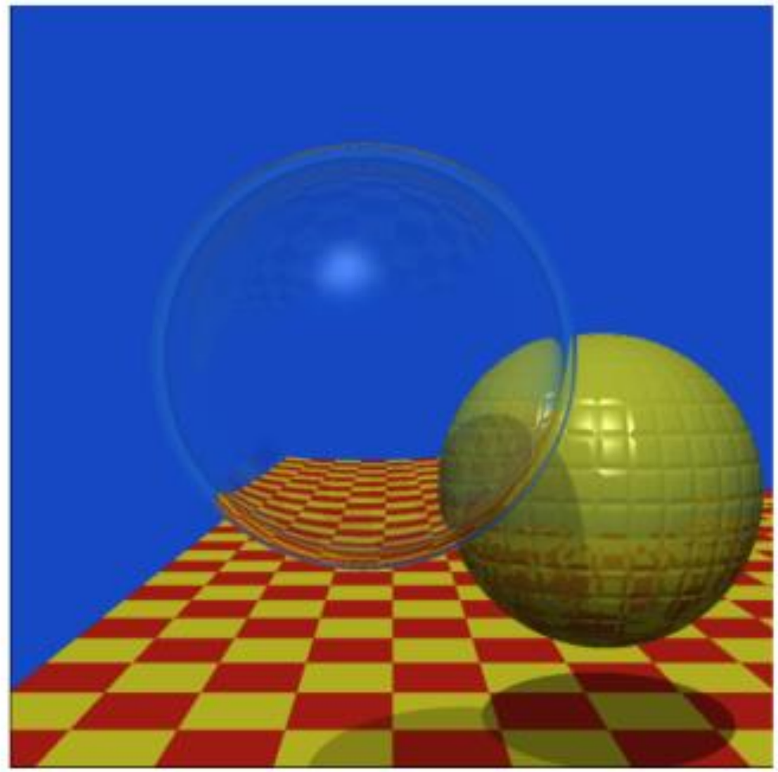
Ray Tracing History

Ray Tracing in Computer Graphics

"An improved Illumination model for shaded display," T. Whitted, CACM 1980

Resolution:
512 x 512

Time:
VAX 11/780 (1979)
74 min.
PC (2006)
6 sec.

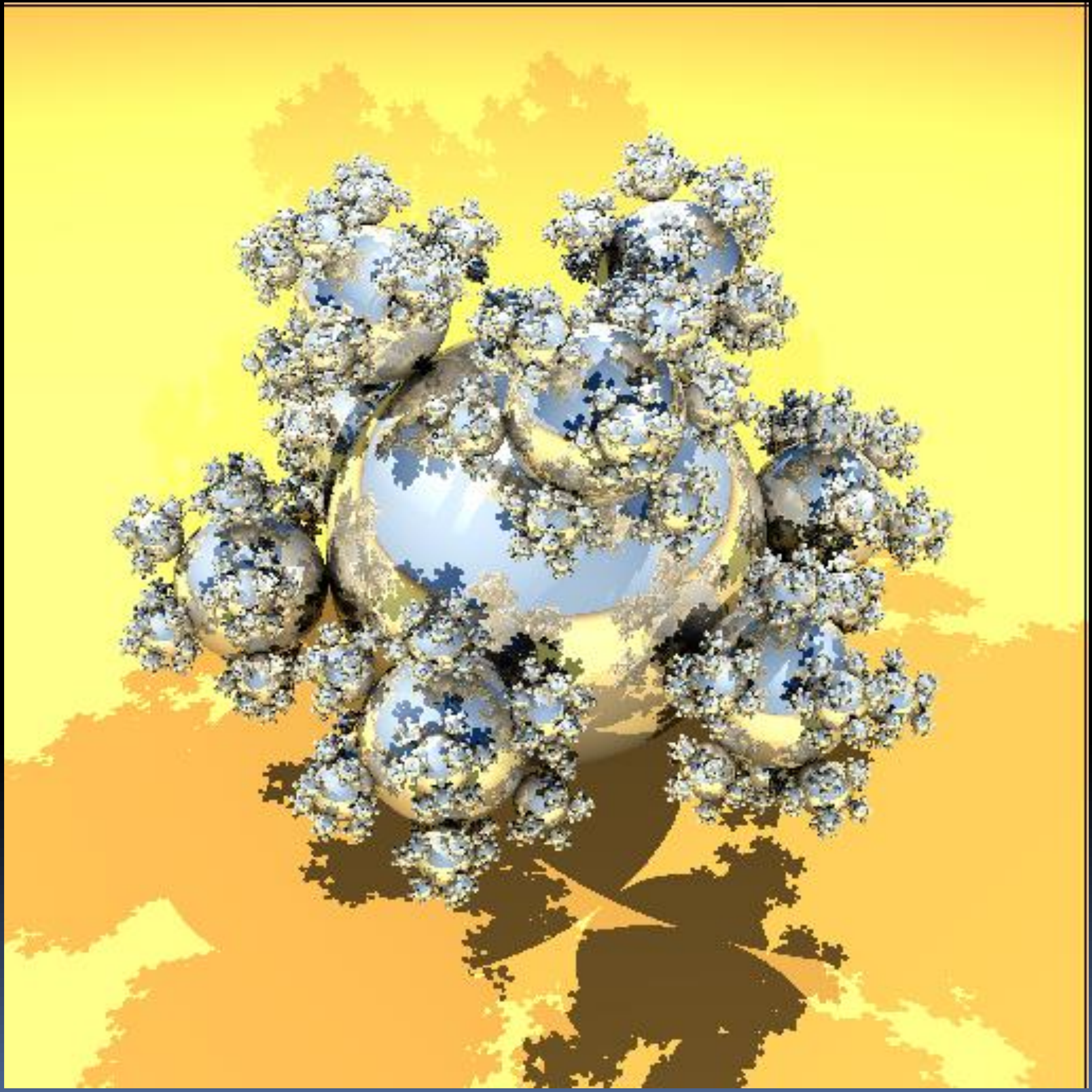


Spheres and Checkerboard, T. Whitted, 1979

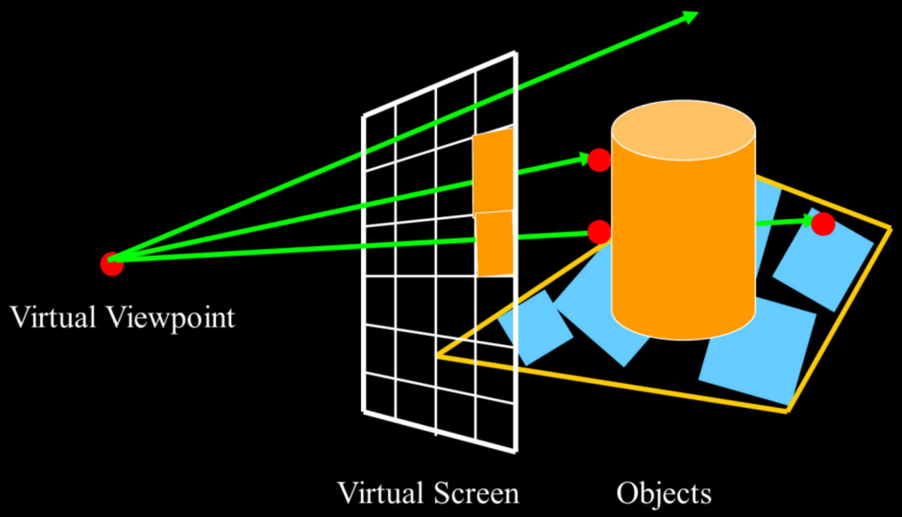
Whitted ray tracing

It combines in a single model:

- Hidden surface removal
- Shading due to direct illumination
- Shading due to indirect illumination (reflection and refraction effects due to mirror/transparent objects)
- Shadow computation (hard shadows)



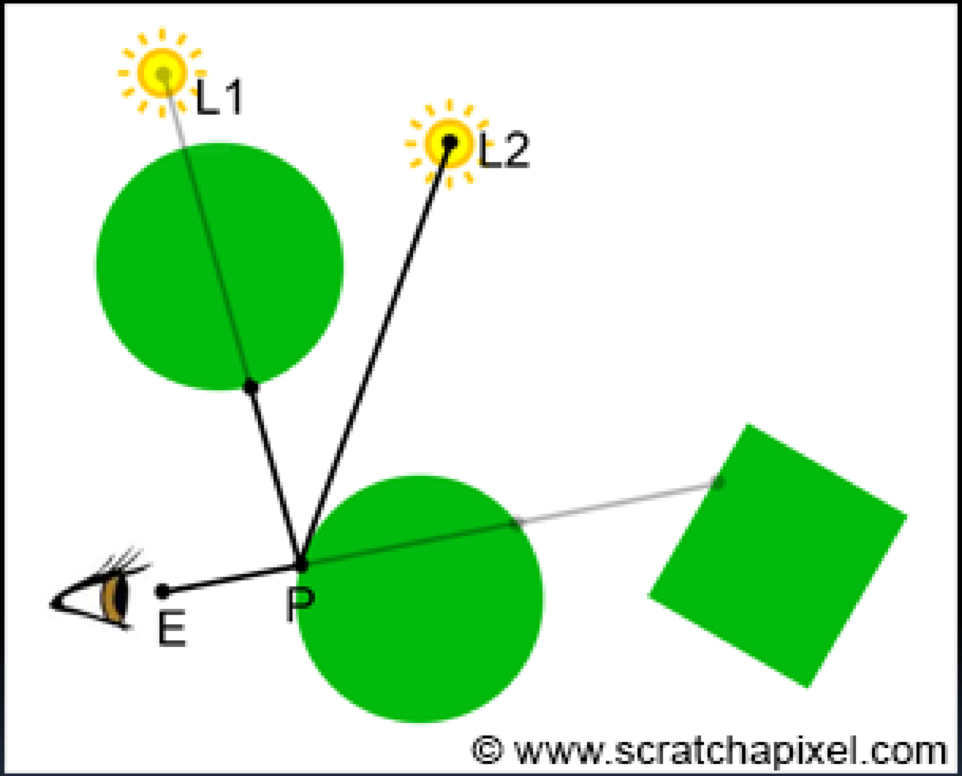
Ray Casting



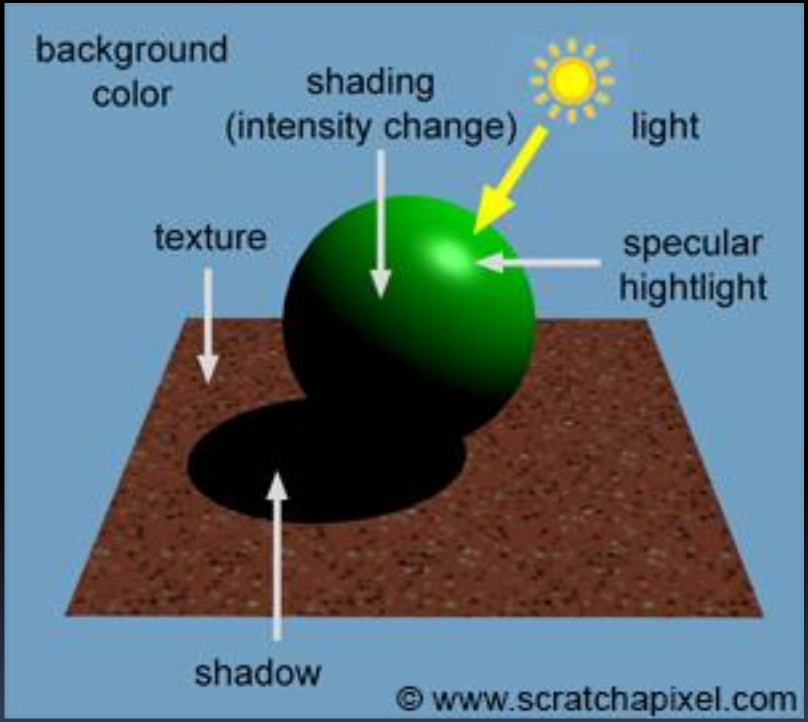

```
For each pixel in the viewport;
   shoot a ray;
   for each object in the scene
       compute intersection ray-object;
       store the closest intersection;

   if there is an intersection
       shade the pixel using color, lights, materials;
   else  /* ray misses all objects */
       shade the pixel with background color
```

Shadows

- Only shades the intersection if not in shadow



Ray Casting



background color
shading (intensity change)
light
texture
specular hightlight
shadow
© www.scratchapixel.com

How does RT differ from Rasterization (slide from CS348b: image synthesis by Matt Pharr)

Primary Visibility: Ray Tracing vs. Rasterization

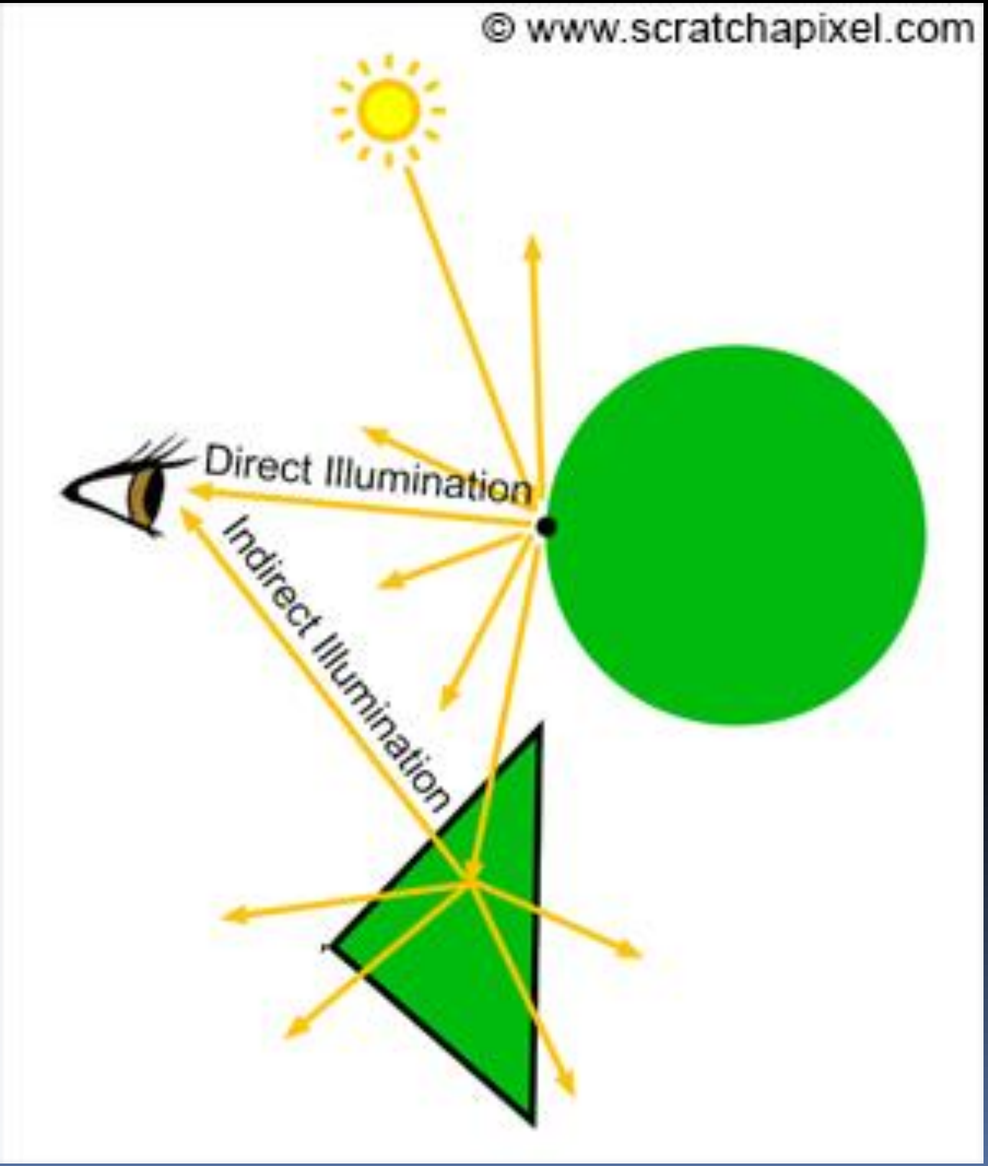

```
for each triangle
  project triangle to image plane
  for each sample
    check sample-in-triangle
    resolve visibility with z-buffer
```

```
for each image sample
  compute corresponding ray
  for each triangle
    check ray-triangle intersection
    record closest intersection
```

- Essentially just a loop interchange...
- Spatial data structures / culling for both so that loops aren't exhaustive

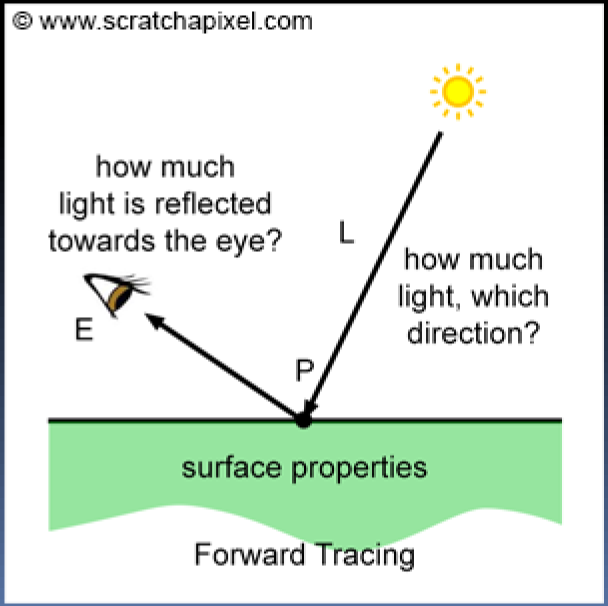
Global Illumination

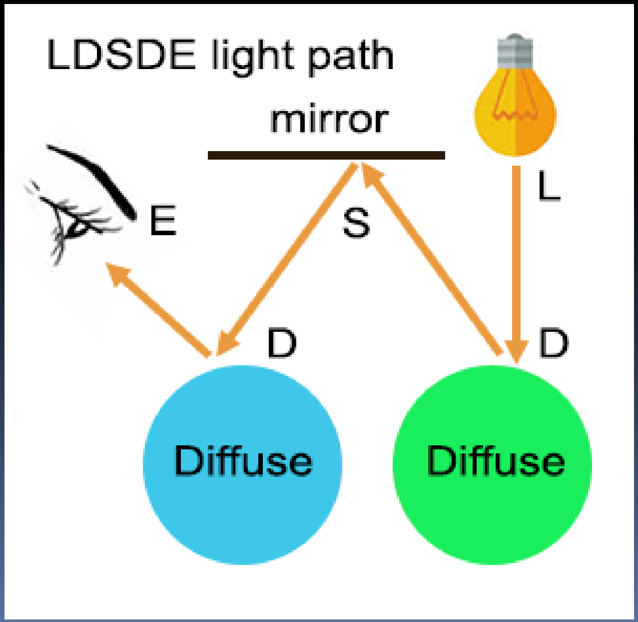
Global Illumination or Light Transport



Light Transport and Shading

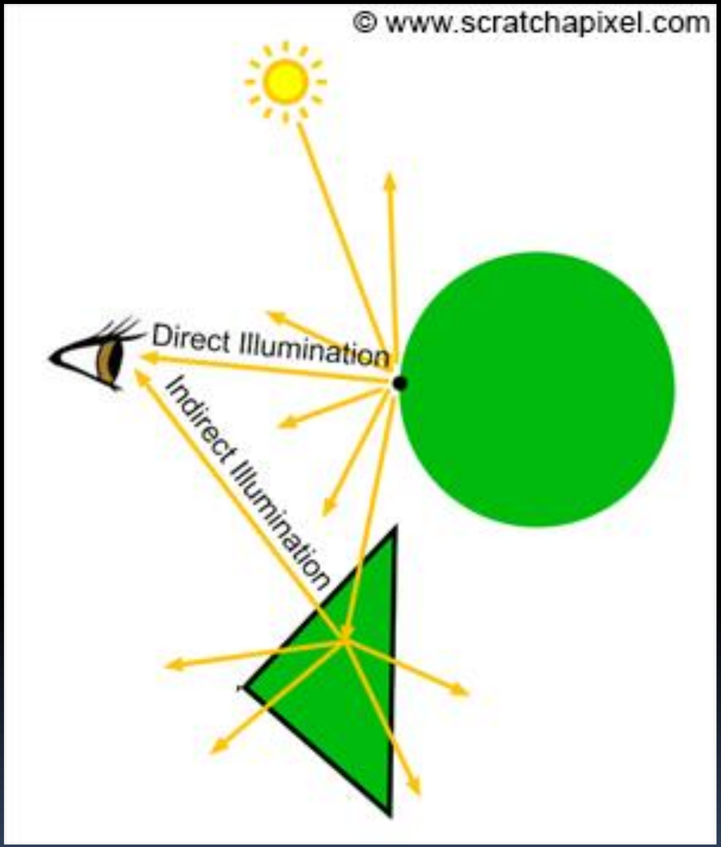
- The appearance of objects, only depends on the way light interacts with matter and travels trough space.
- Shading: Interaction light-matter
- Light transport: determine and follow path light rays due to inter-reflections

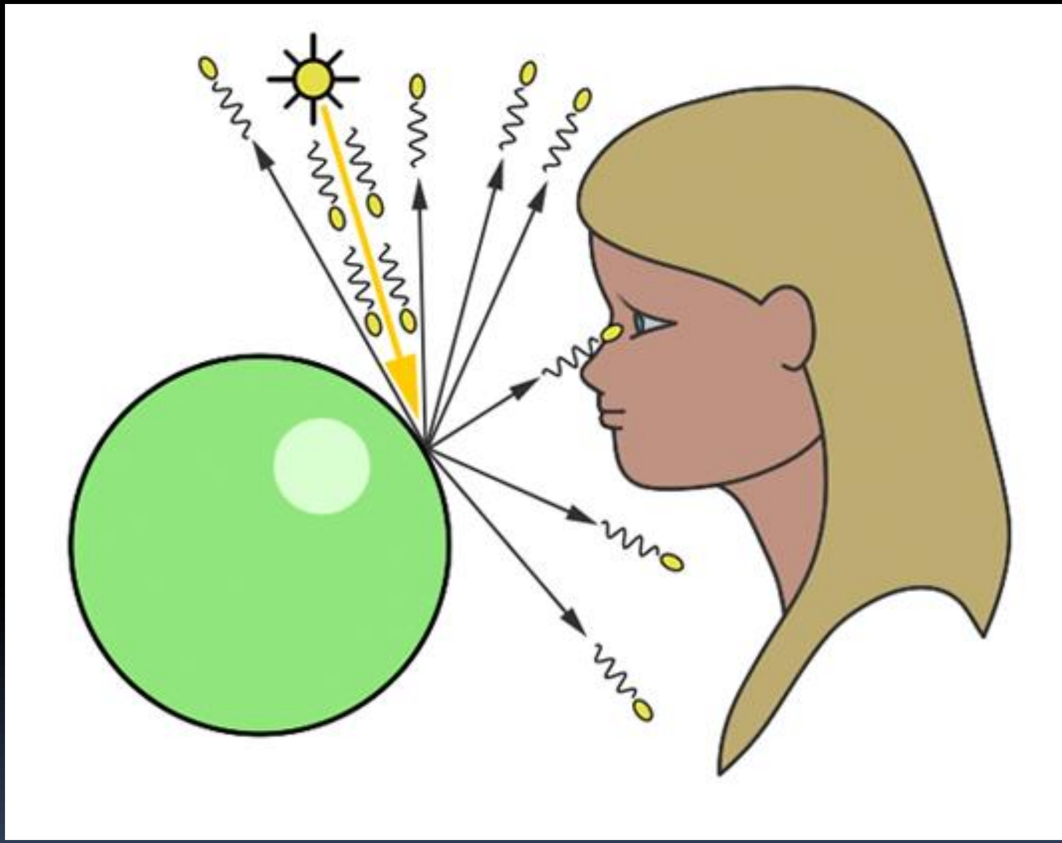




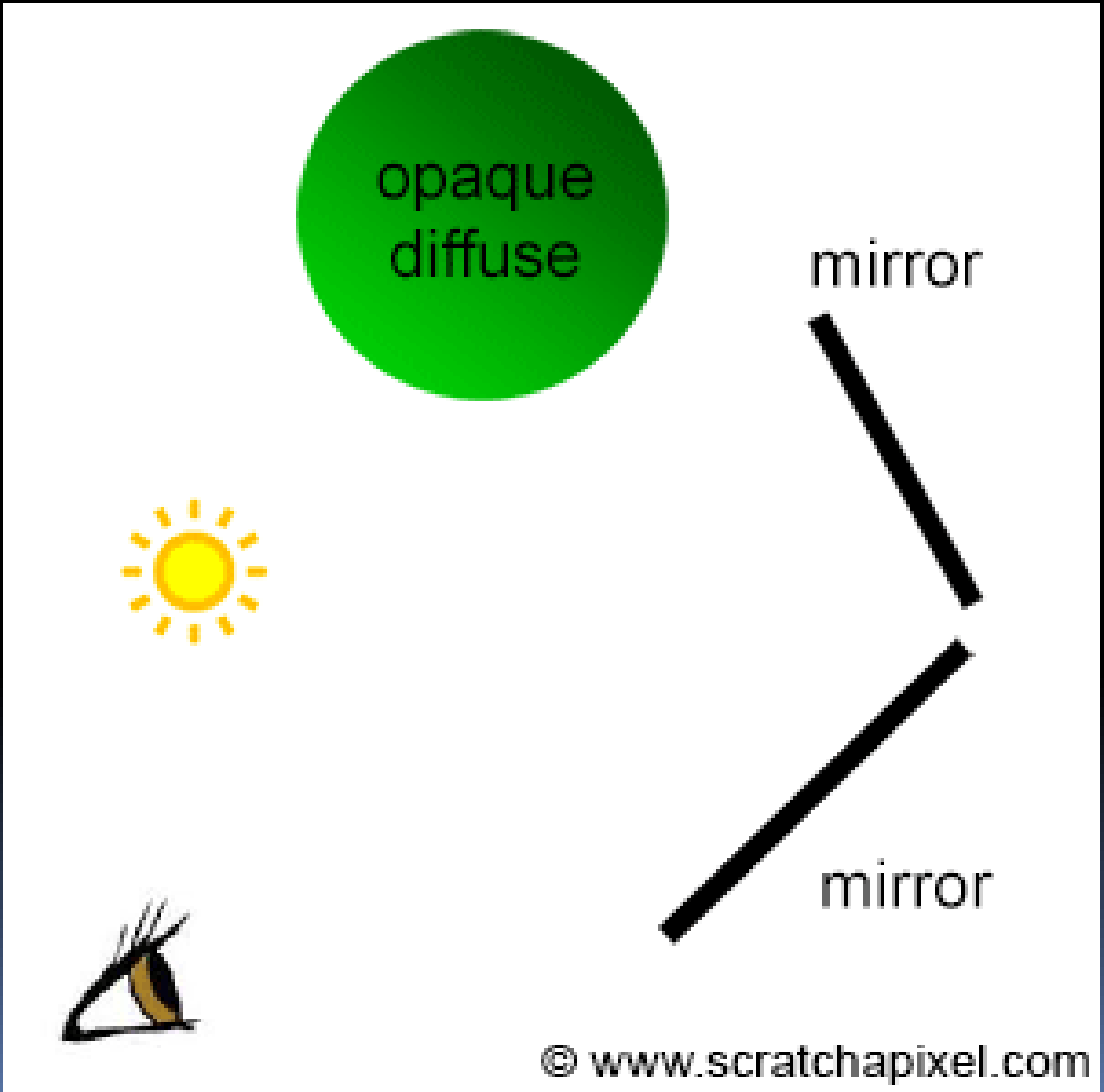
Forward tracing

- aka Light Tracing

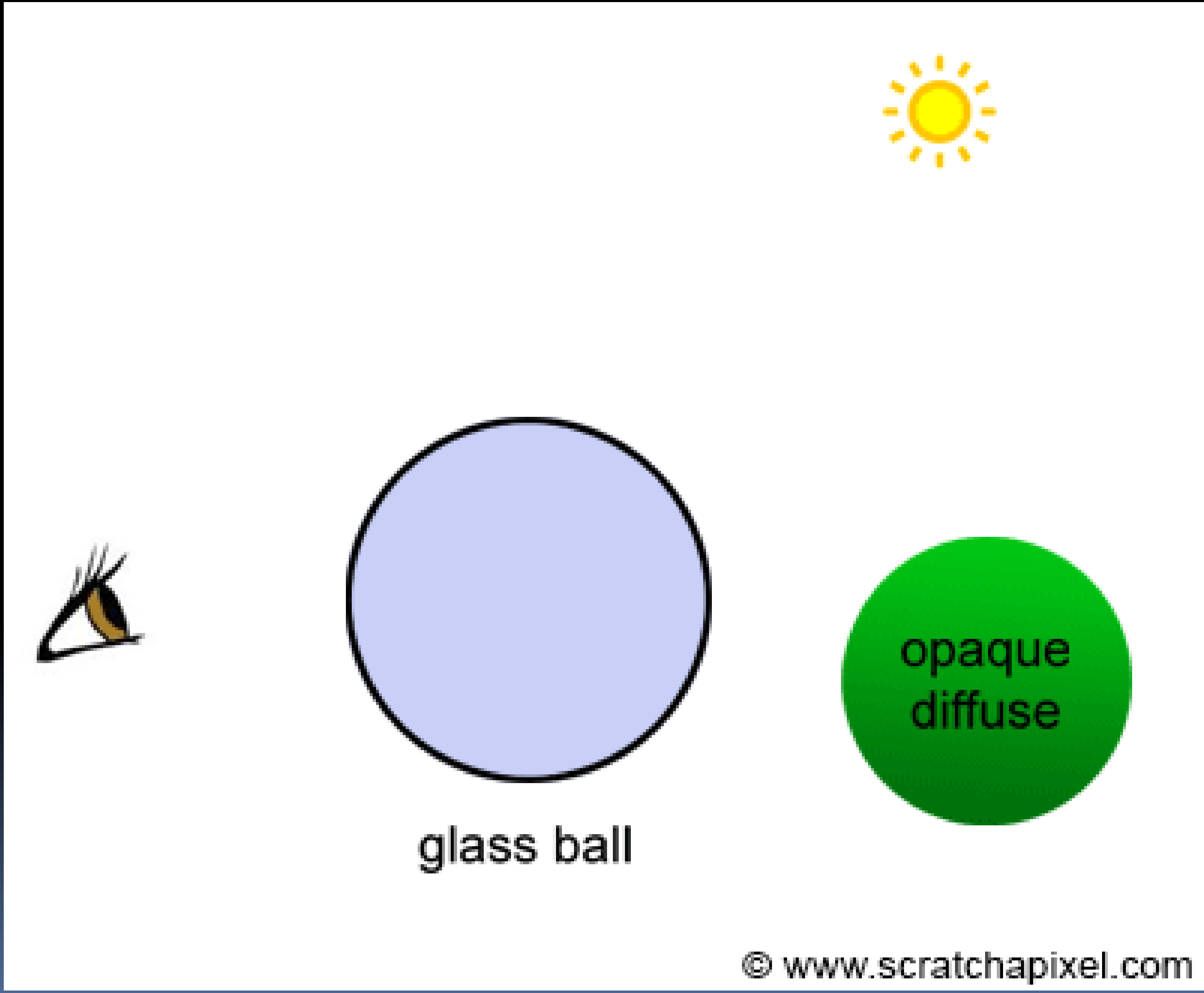




Backward Tracing



Backward Tracing



Ray-Tracing de Turner Whitted

- Backward Ray Tracer
- We trace light rays from the eye through a pixel in the viewport - **primary rays**
- ie we follow light beams in the reverse direction of the light propagation
- Find the intersection with the nearest object during the backwards trace of the ray
- The color of the ray (hence at the required pixel) is made up of 3 contributions:

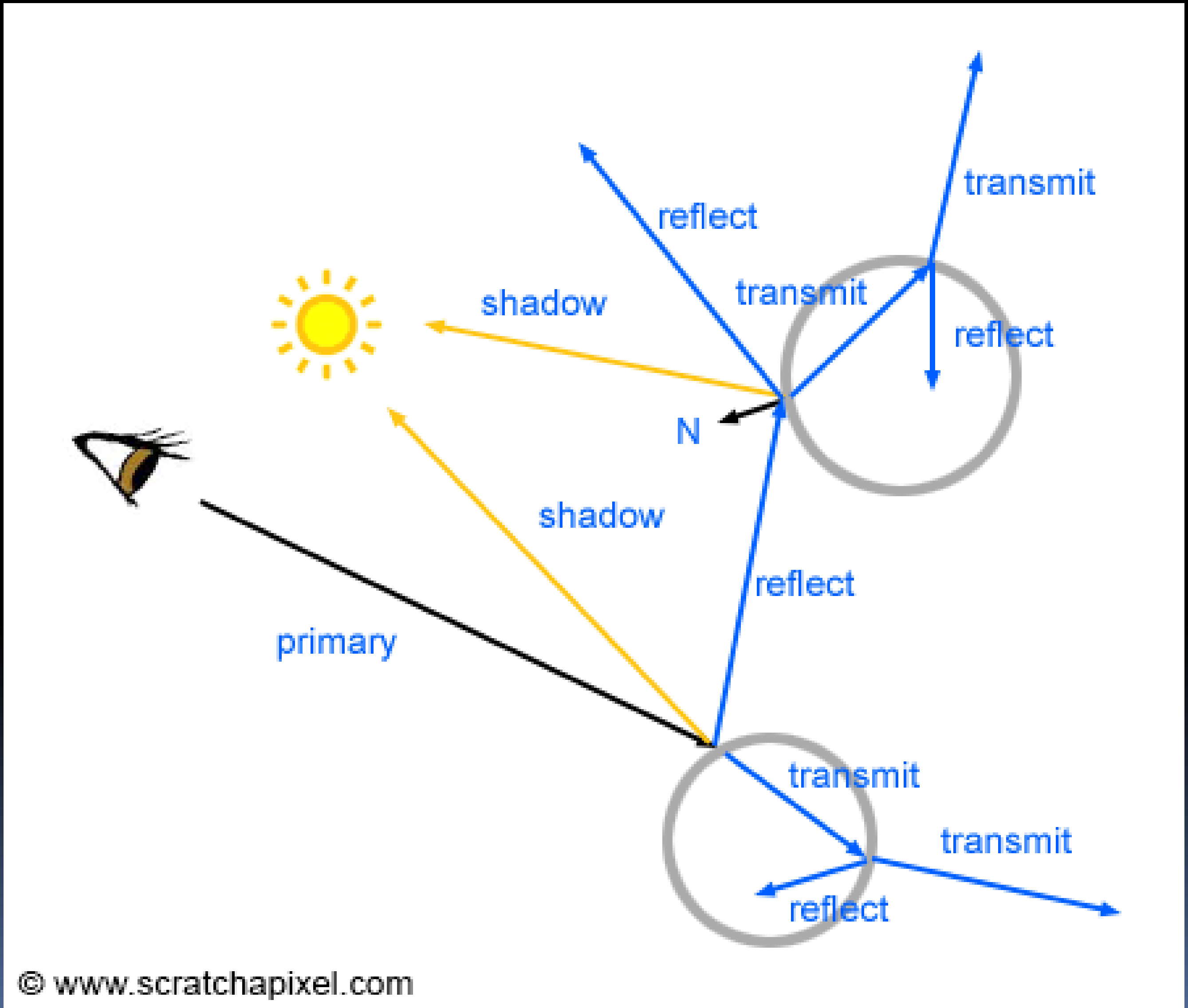
  -local color due to direct illumination (it can be in shadow);

  -color from a ray coming from the reflection direction – reflected ray;

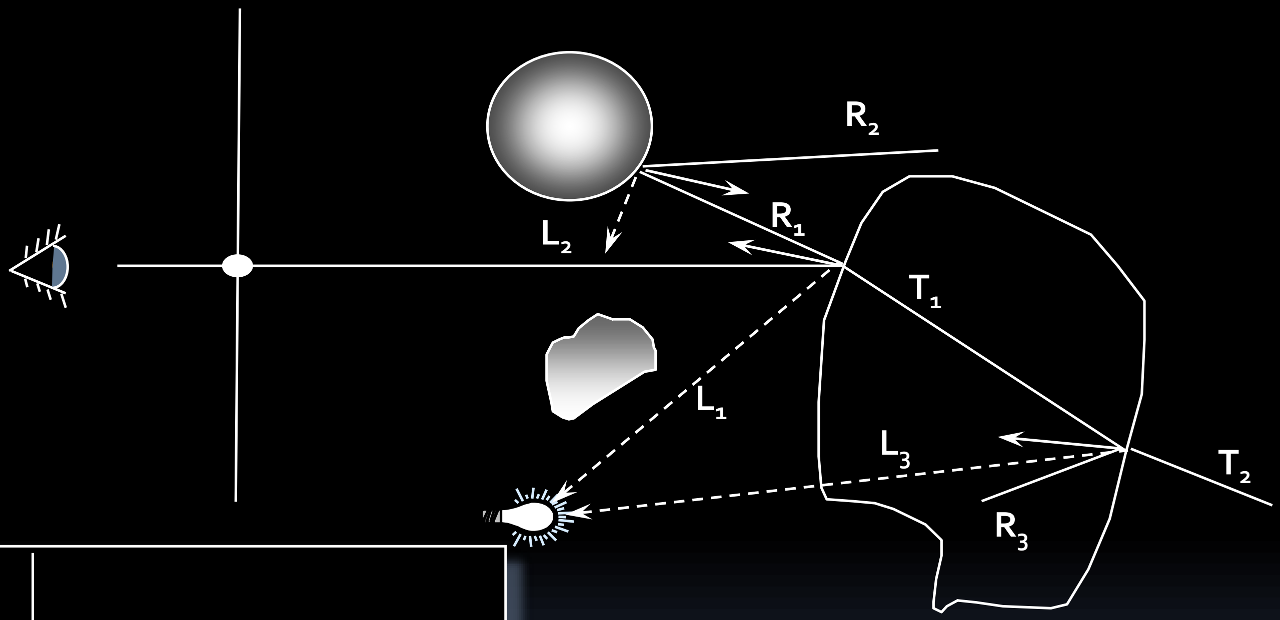
  - color from a ray coming from the refraction direction – transmitted or refracted ray;
- Shadow feelers, reflected and refracted rays are called **secondary rays**

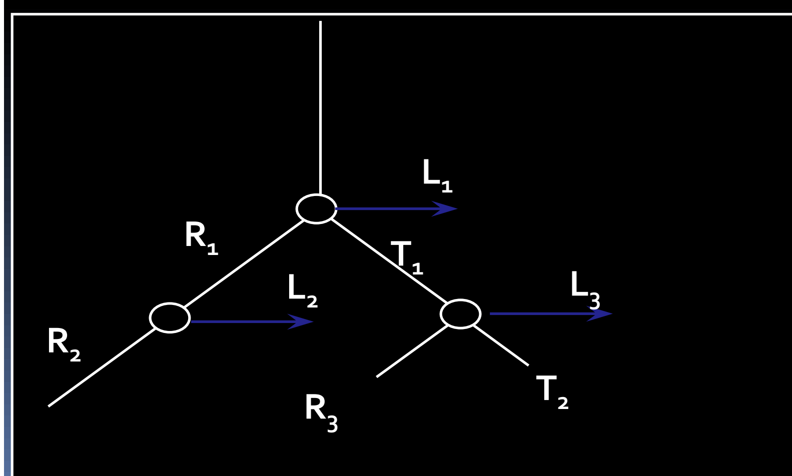
```
Define the viewpoint, the view window  and the viewport resolution
for each pixel in the viewport
{
        compute a ray in World space from the eye towards  the pixel;
        pixel_color = trace ( scene, eye, primary ray direction, 1);
}
```

Recursive nature

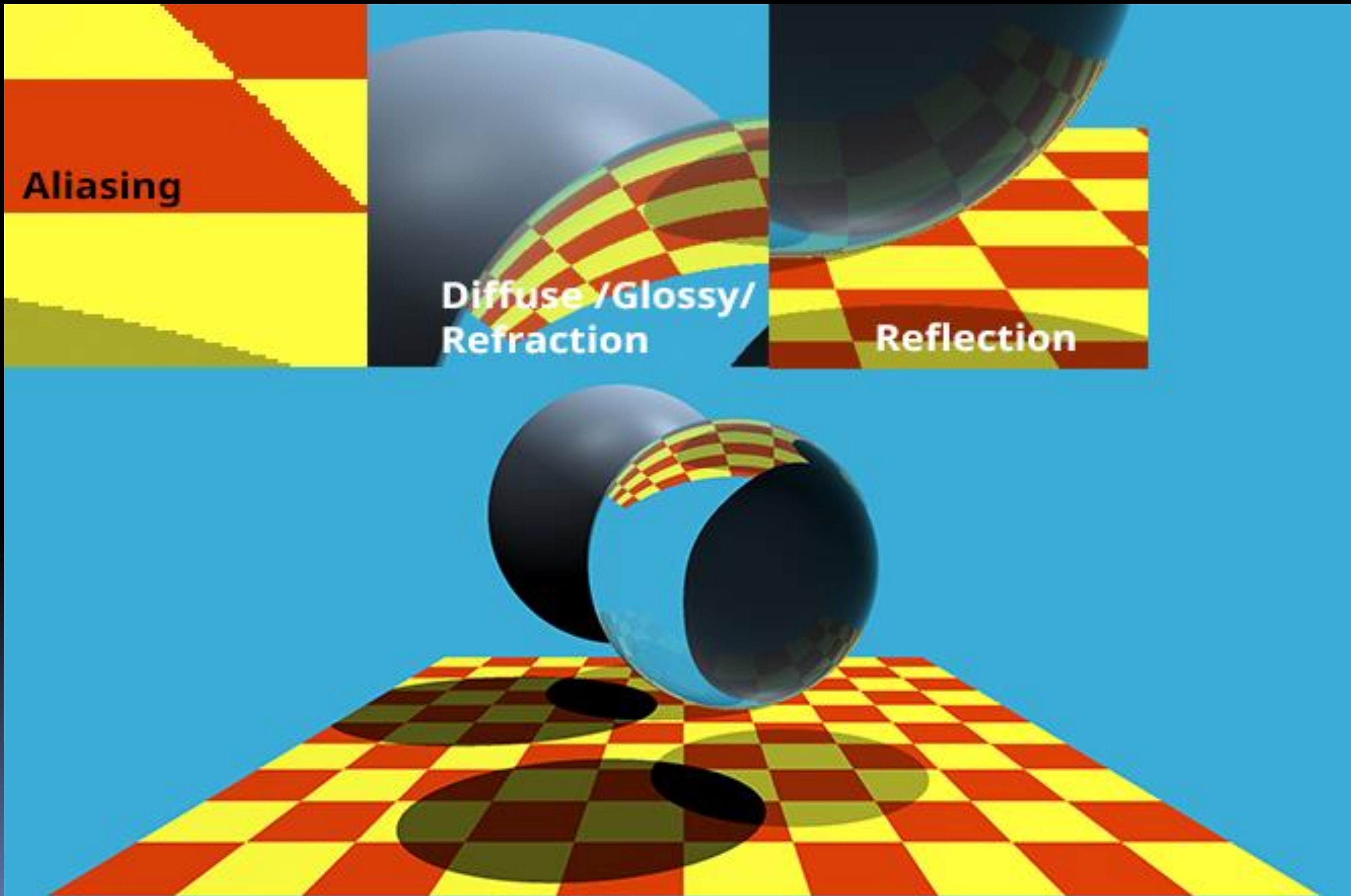


Algorithm’s recursive nature



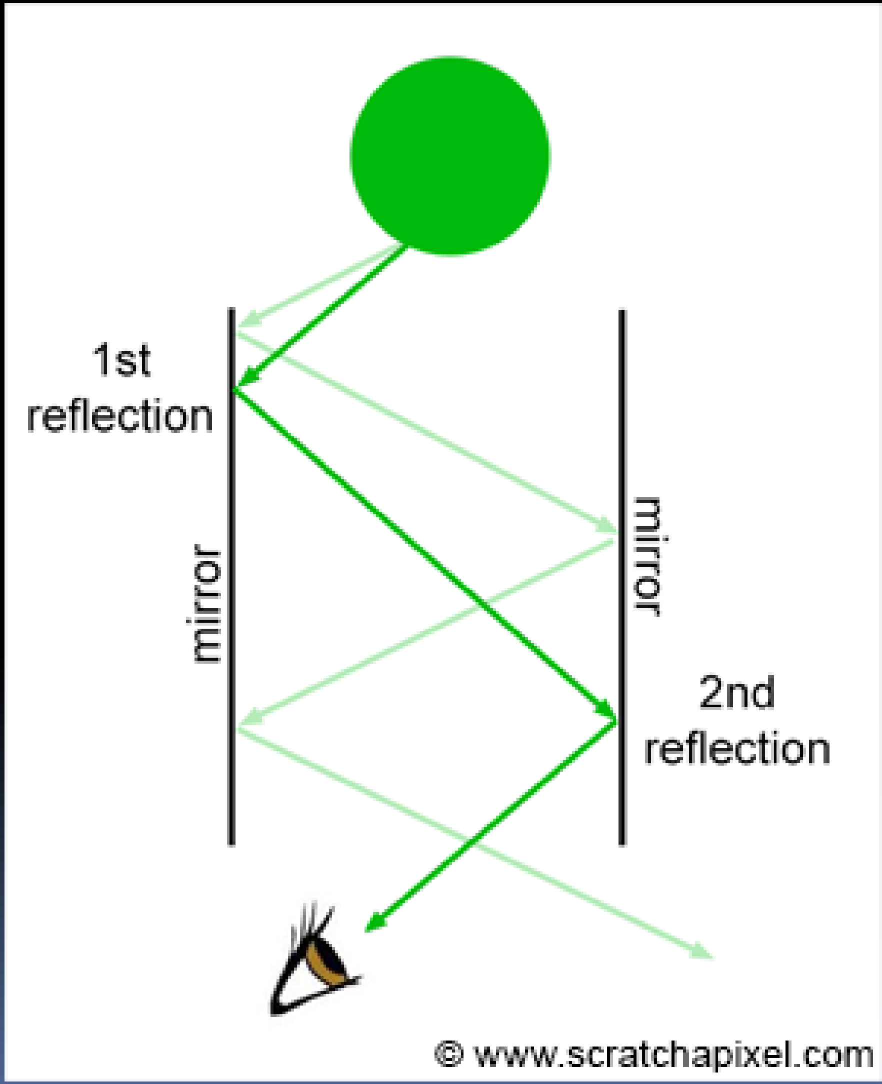


Result



Mirror Maze

Mirror Maze



1st
reflection
mirror
mirror
2nd
reflection
© www.scratchapixel.com

```
Color trace (Scene scene, Vector3d origin, Vector3d ray direction, int depth)
{
        intersect ray with all objects and find a hit point (if any) closest to the start of the ray
        if (!intersection point) return BACKGROUND;
        else {
                compute normal at the hit point;
                for (each source light) {
                        L = unit light vector from hit point to light source;
                         if (L • normal>0)
                                if (!point in shadow); //trace shadow ray
                                        color = diffuse color + specular color;
                }
                if (depth >= maxDepth) return color;

                if (reflective object) {
                         rRay = calculate ray in the reflected direction;
                         rColor = trace(scene, point, rRay direction, depth+1);
                         reduce rColor by the specular reflection coefficient and add to color; }

                if (transparent object) {
                   tRay = calculate ray in the refracted direction;
                   tColor = trace(scene, point, tRay direction, depth+1);
                  reduce tColor by the transmittance coefficient and add to color; }

                return color;
                }
```

Rays

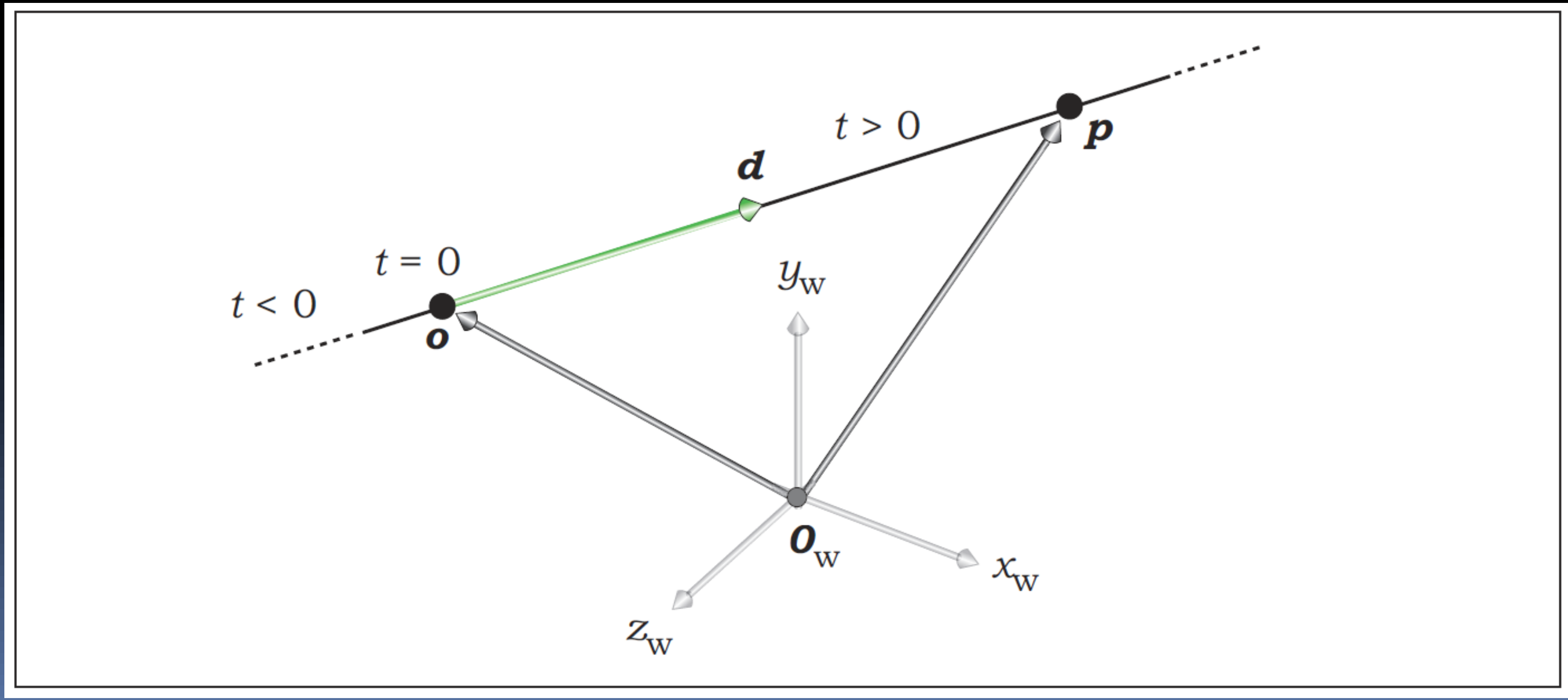
Parametric formulation: $\boldsymbol{p = o + td}$

p: a point on the ray

o: origin of the ray

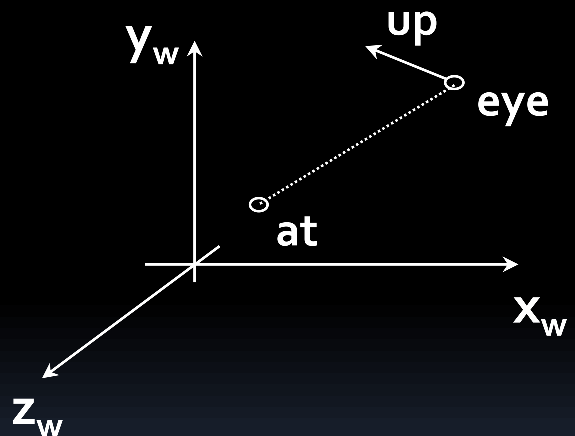
t: scalar parameter

d: unit vector giving the direction of the ray

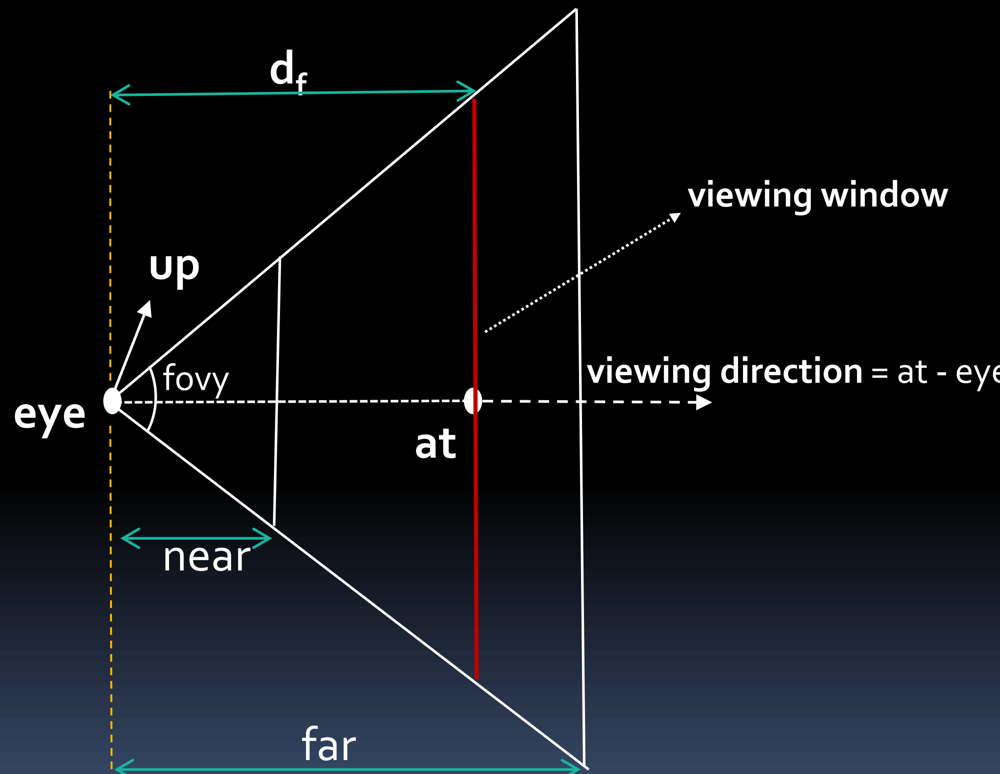


Camera Position and Orientation

eye = viewer
at = target point in the center of **viewing window** (near plane in OpenGL)
up = up direction
d_f (view distance) = ||at – eye||

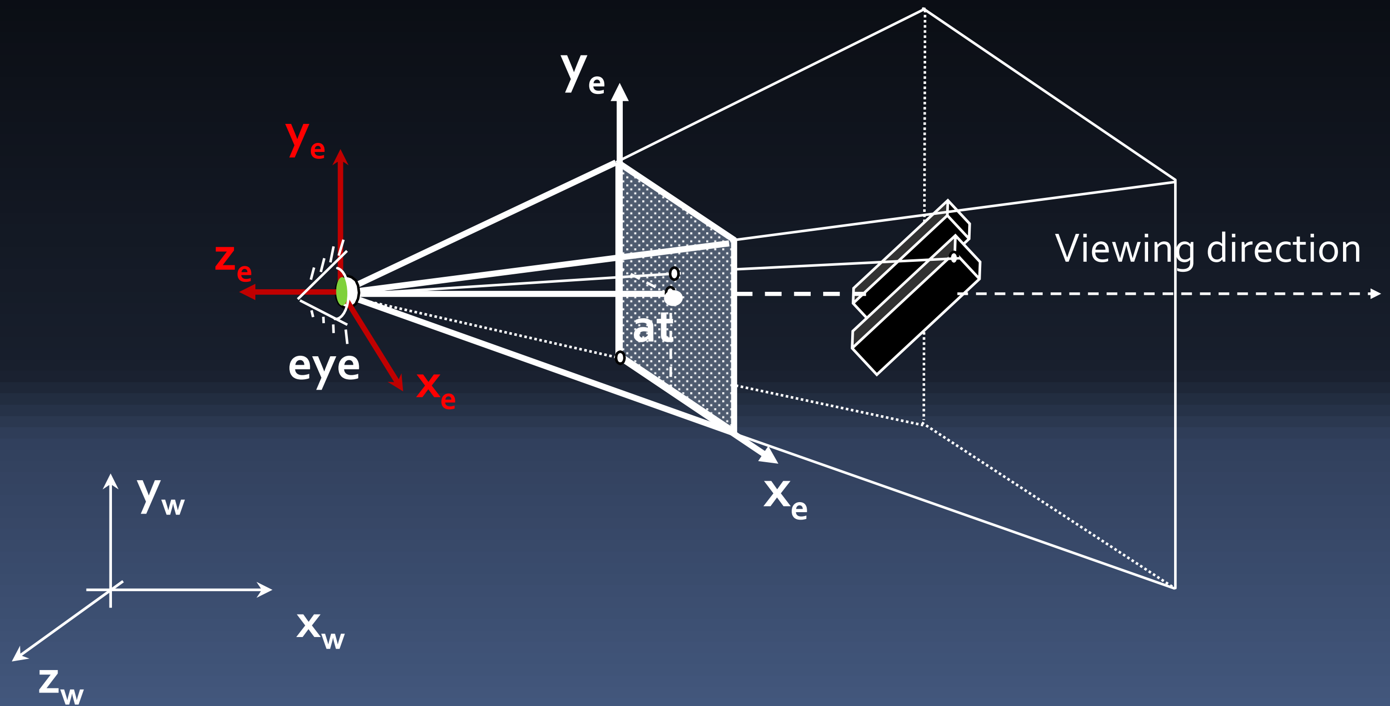


World Coordinates



The axes in the Camera Frame

$x_e\, y_e\, z_e$ (aka $u\ v\ {-n}$)



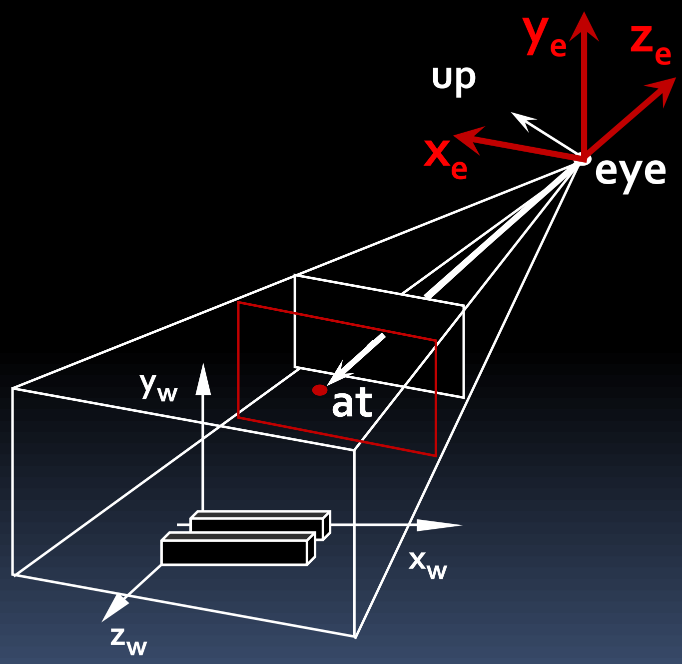
Camera Frame - $x_e y_e z_e$

data:
eye, at, up

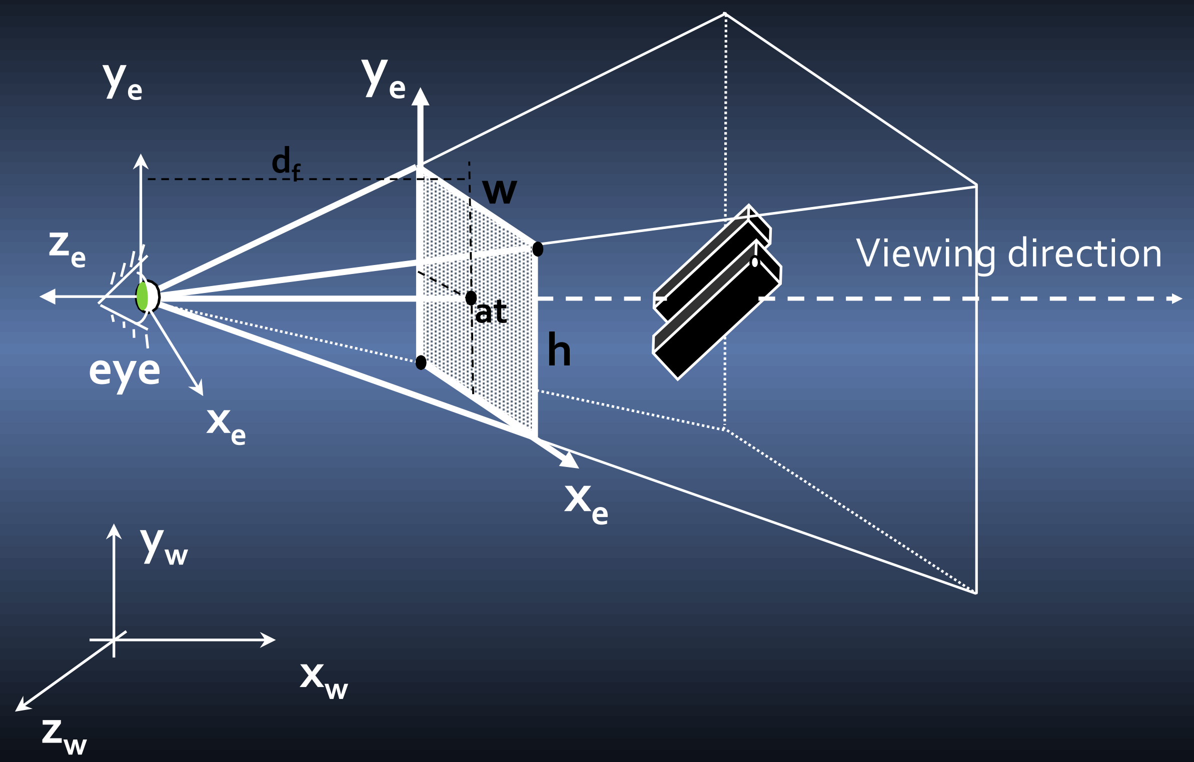
$$\mathbf{z}_e = \frac{1}{\|\mathbf{eye} - \mathbf{at}\|}(\mathbf{eye} - \mathbf{at})$$

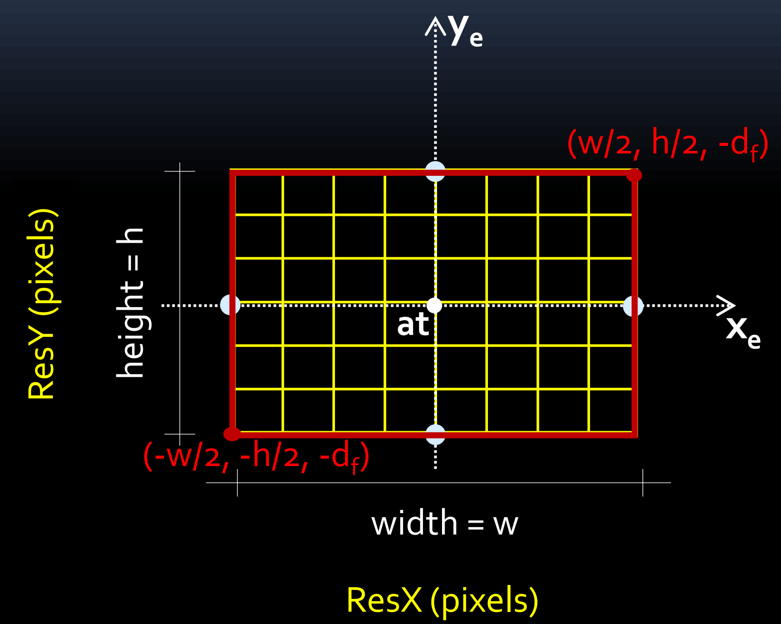
$$\mathbf{x}_e = \frac{1}{\|\mathbf{up} \times \mathbf{z}_e\|}(\mathbf{up} \times \mathbf{z}_e)$$

$$\mathbf{y}_e = \mathbf{z}_e \times \mathbf{x}_e$$

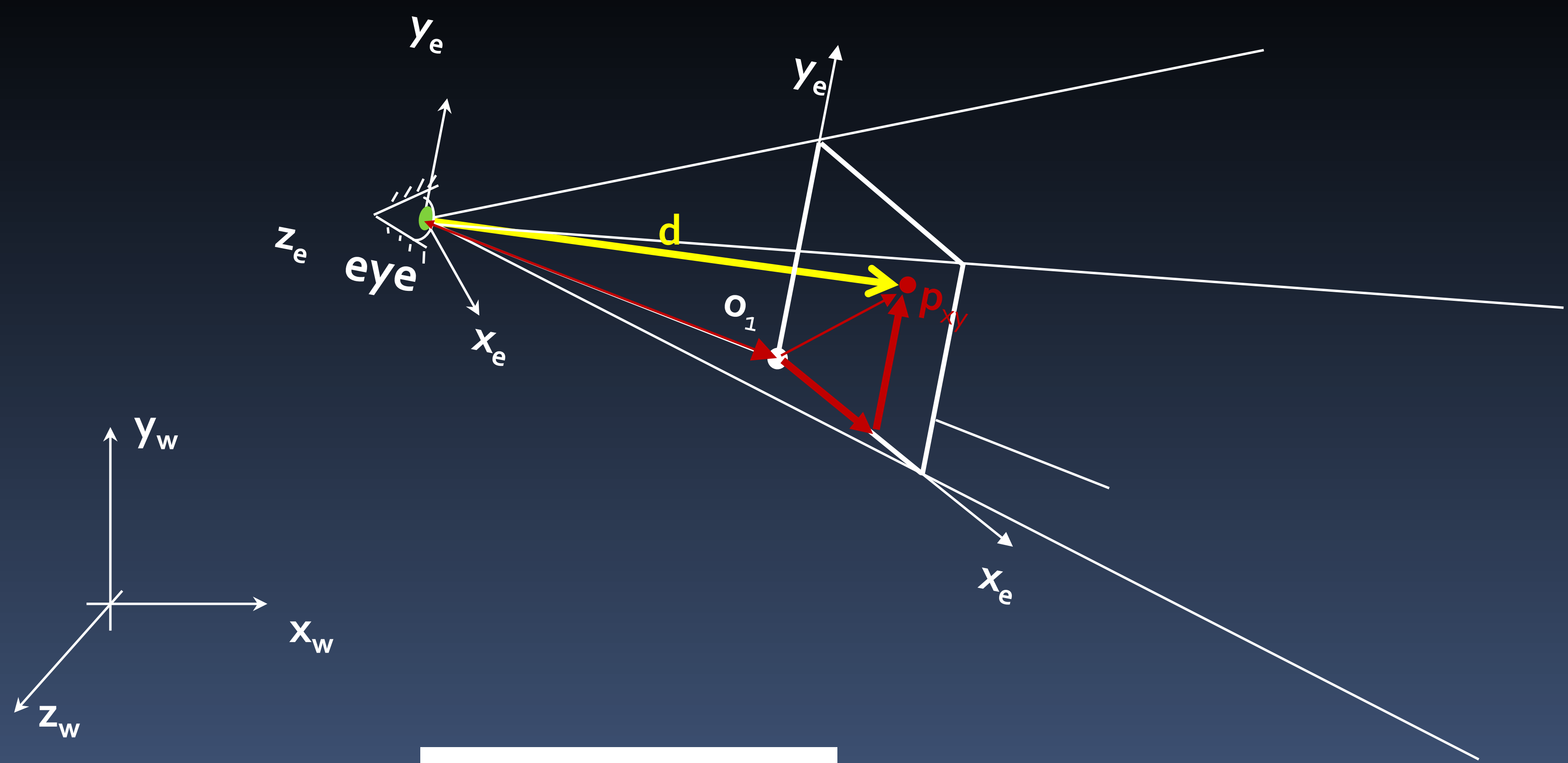


Viewing window and viewport





Primary Rays (World Coordinates)



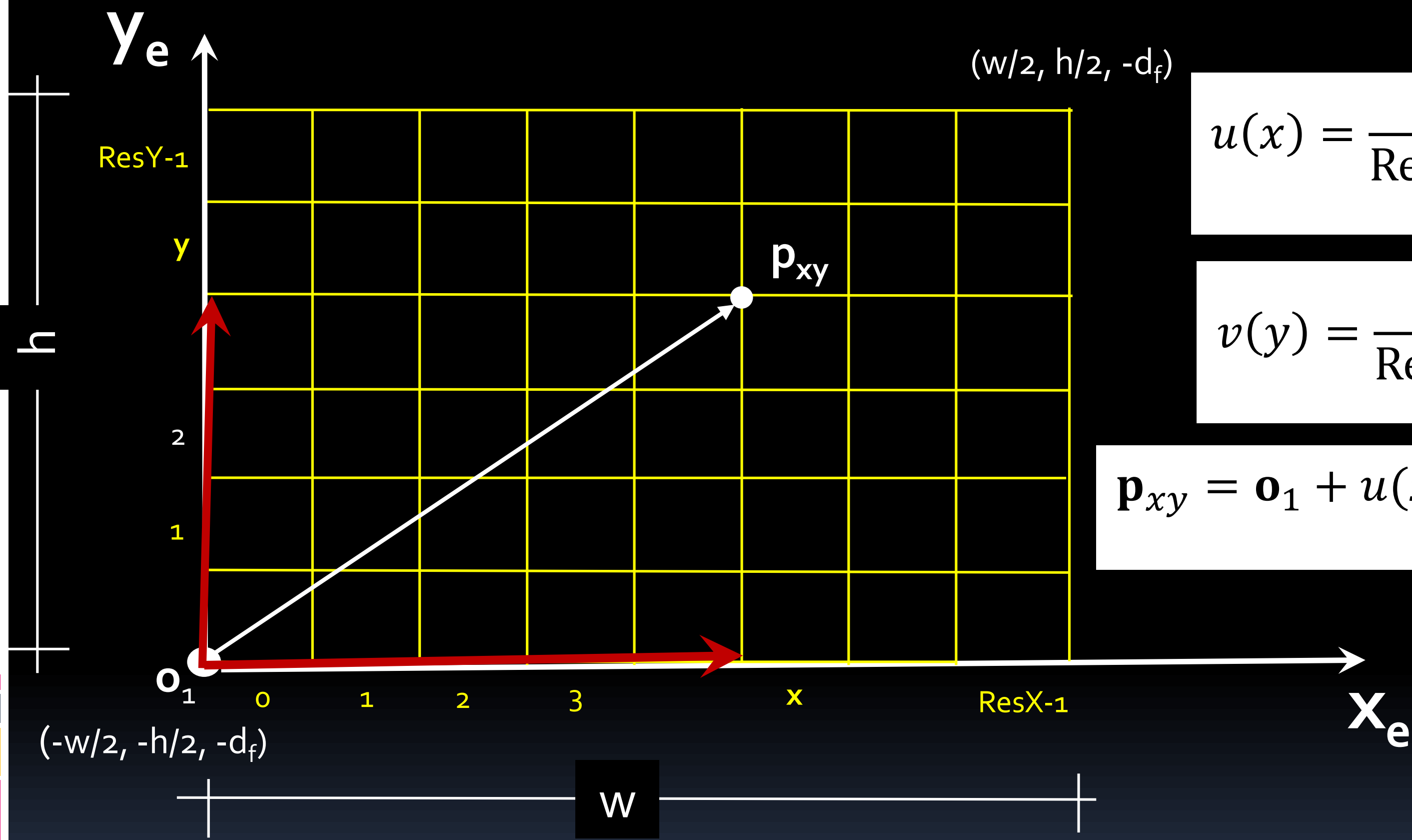
$$ray: \quad \mathbf{p}(t) = \mathbf{o} + t\mathbf{d}$$

$$\mathbf{o} = \mathbf{eye} \; and \; \mathbf{d} = \frac{\mathbf{p}_{xy} - \mathbf{eye}}{|\mathbf{p}_{xy} - \mathbf{eye}|}$$

$$\mathbf{p}_{xy} = \mathbf{o}_1 + u(x)\hat{\mathbf{x}}_e + v(y)\hat{\mathbf{y}}_e$$

Computing Primary Rays

Ray at the left-bottom corner of the unit square pixel



$$u(x) = \frac{w}{\mathrm{Re}\,s\,X}\,x$$

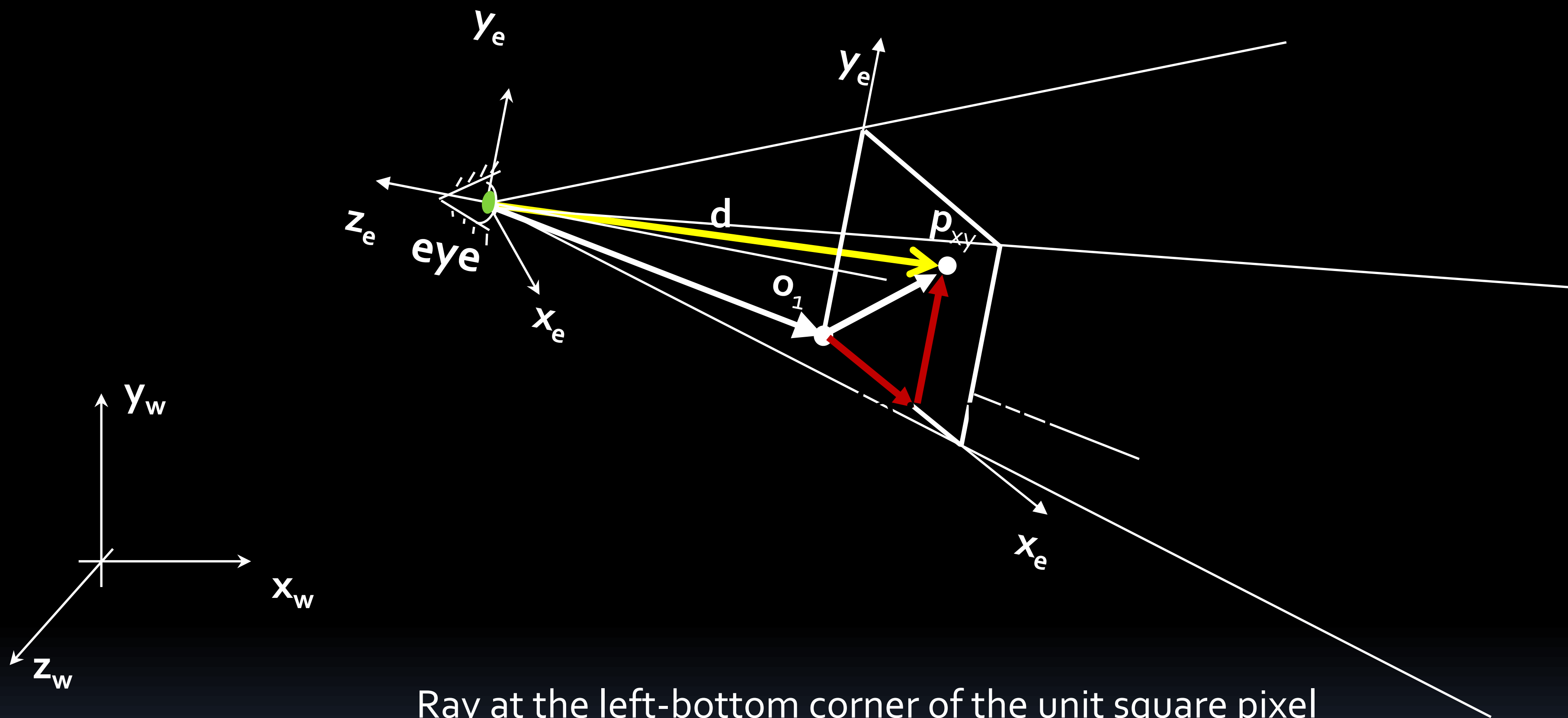
$$v(y) = \frac{h}{\mathrm{Re}\,s\,Y}\,y$$

$$\mathbf{p}_{xy} = \mathbf{o}_1 + u(x)\hat{\mathbf{x}}_e + v(y)\hat{\mathbf{y}}_e$$

$$\mathbf{p}_{xy} = \mathbf{o}_1 + w\frac{x}{\mathrm{Re}\,s\,X}\hat{\mathbf{x}}_e + h\frac{y}{\mathrm{Re}\,s\,Y}\hat{\mathbf{y}}_e$$

$$d_f = \left\|\mathbf{eye} - \mathbf{at}\right\|$$

Primary Rays



Ray at the left-bottom corner of the unit square pixel

$$\mathbf{o}_1 = eye - \frac{w}{2}\hat{\mathbf{x}}_e - \frac{h}{2}\hat{\mathbf{y}}_e - d_f\hat{\mathbf{z}}_e \quad and \ \mathbf{d} = \frac{\mathbf{p}_{xy} - \mathbf{eye}}{|\mathbf{p}_{xy} - \mathbf{eye}|}$$

Putting all together:

$$\mathbf{d} = normalize(w\left(\frac{x}{\mathrm{Re}\, sX} - \frac{1}{2}\right)\hat{\mathbf{x}}_e + h\left(\frac{y}{\mathrm{Re}\, sY} - \frac{1}{2}\right)\hat{\mathbf{y}}_e - d_f\hat{\mathbf{z}}_e)$$

Camera Data in C

```
struct _Camera {
  /* Camera definition*/
  Vector eye, at, up;
  float  fovy;
  float  near,far; //hither and yon planes
  int    ResX,ResY;

  float  w,h;
  Vector xe,ye,ze; //uvn frame
};

typedef struct _Camera  Camera;
```

```
Camera* camCreate( Vector eye, Vector at, Vector up,
       double fovy, double near, double far, int ResX, int ResY );
```

```
Ray camGetPrimaryRay( Camera camera, double x, double y );
```

The Camera object

Initialization:

Data input: fov, ResX, ResY, near, far, **eye**, **at**, **up**

$$d_f = \|\mathbf{eye} - \mathbf{at}\|$$

$$h = 2 d_f \tan\left(\frac{fov}{2}\right)$$

$$w = \frac{\mathrm{Re}\, sX}{\mathrm{Re}\, sY} h$$

$$\mathbf{z}_e = \frac{1}{\|\mathbf{eye} - \mathbf{at}\|}(\mathbf{eye} - \mathbf{at})$$

$$\mathbf{x}_e = \frac{1}{\|\mathbf{up} \times \mathbf{z}_e\|}(\mathbf{up} \times \mathbf{z}_e)$$

$$\mathbf{y}_e = (\mathbf{z}_e \times \mathbf{x}_e)$$

Ray in parametic form : **o + td** (normalize **d**; why?)

Given: x, y

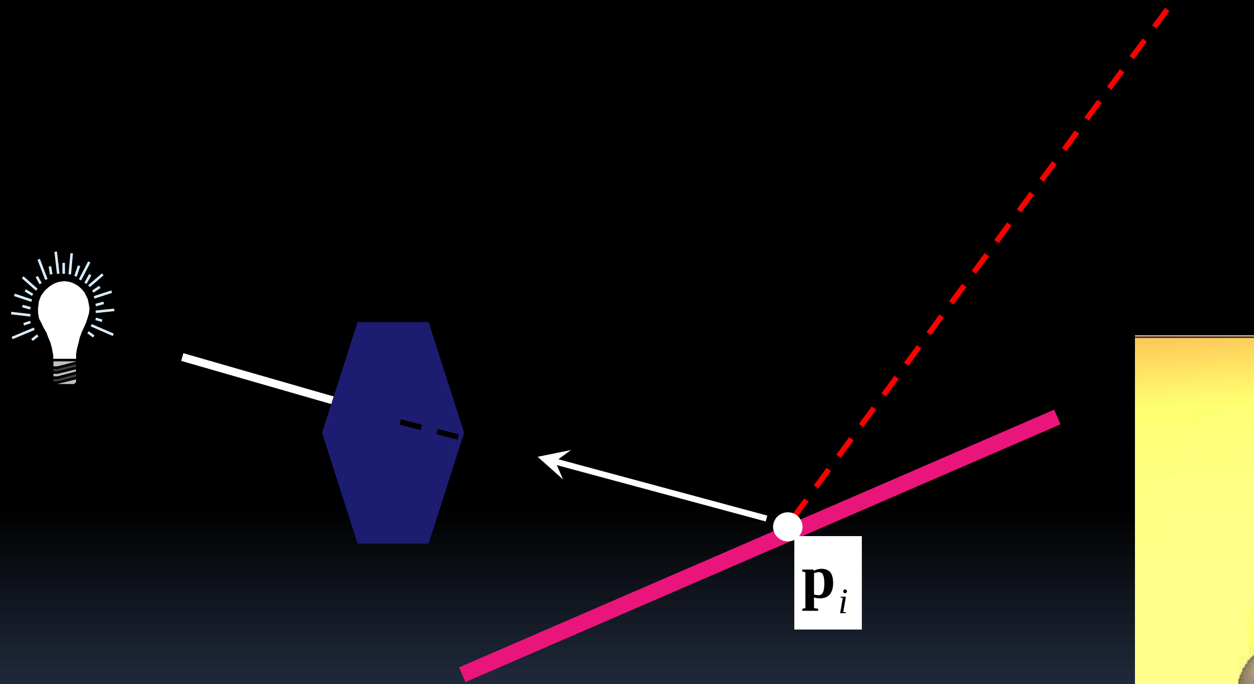
Ray at the center of the square pixel

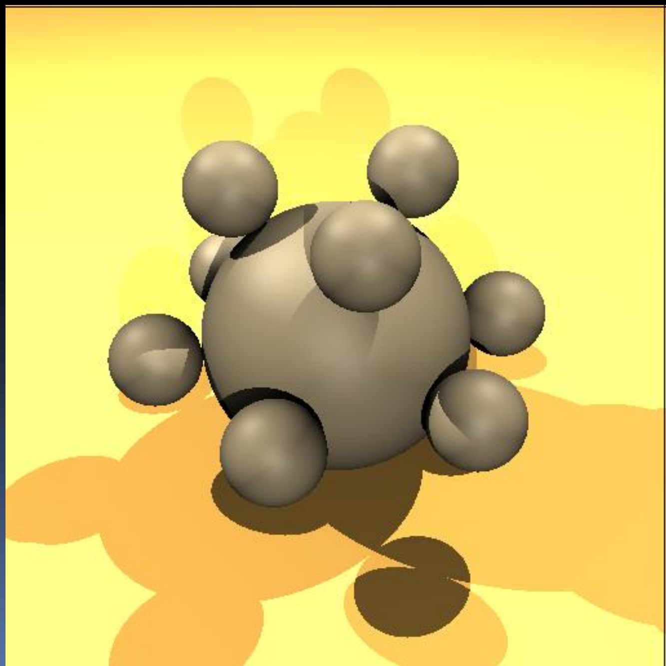
$$\mathbf{o} = \mathbf{eye}$$

$$\mathbf{d} = normalize\left(w\left(\frac{x+0.5}{\mathrm{Re}\ sX} - \frac{1}{2}\right)\hat{\mathbf{x}}_e + h\left(\frac{y+0.5}{\mathrm{Re}\ sY} - \frac{1}{2}\right)\hat{\mathbf{y}}_e - d_f \hat{\mathbf{z}}_e\right)$$

Shadow Feeleers

$$\text{Shadow ray}: \mathbf{p}(t) = \mathbf{p}_i + t\,\hat{\mathbf{r}}_s$$

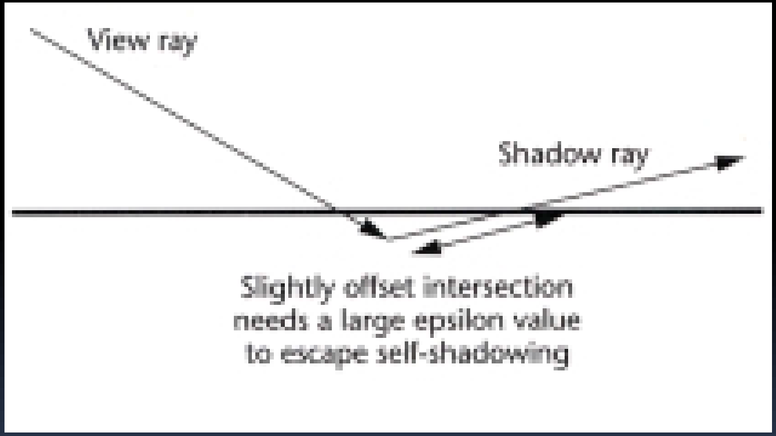




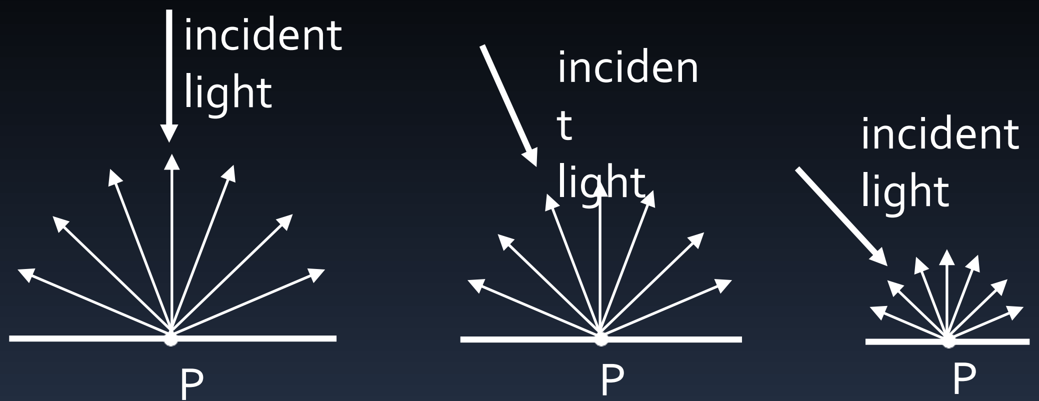
No light at that point

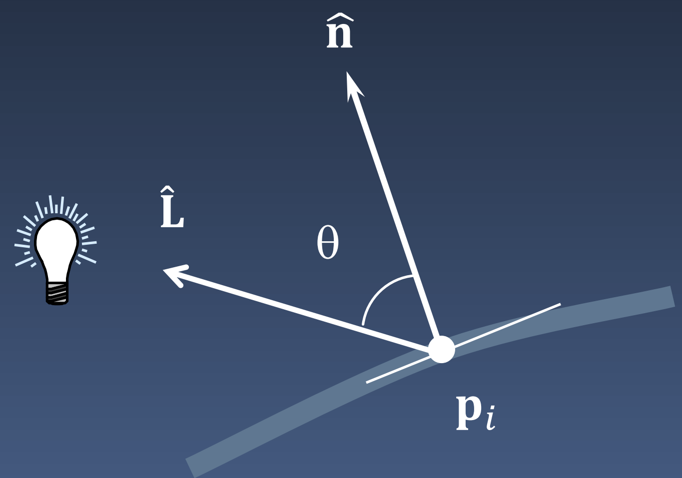
Robust intersections computation

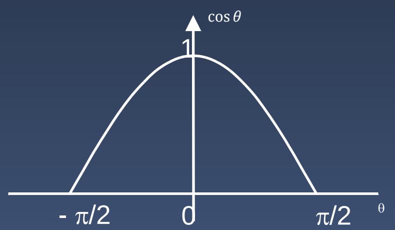
- Read the slides **“Geometry Intersections”**
- Self-intersections problem due to floating-point precision (secondary rays and shadow-feelers)
- Solution: Slightly offset intersections
- Andrew Woo et al., “It’s Really Not a Rendering Bug, You See…”, IEEE CG&A, September 1996, vol. 21. Not a good solution! Why?



Diffuse (lambert) Reflection

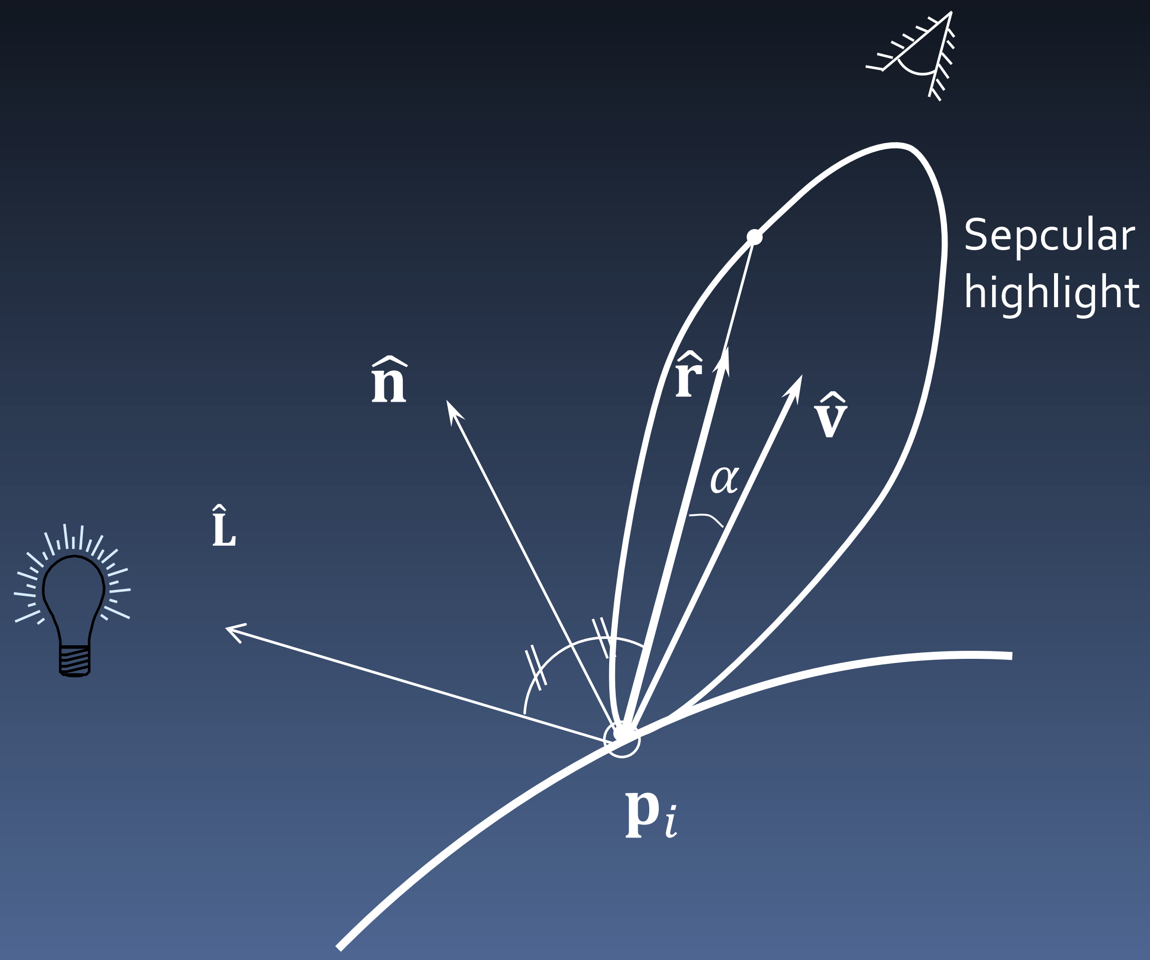






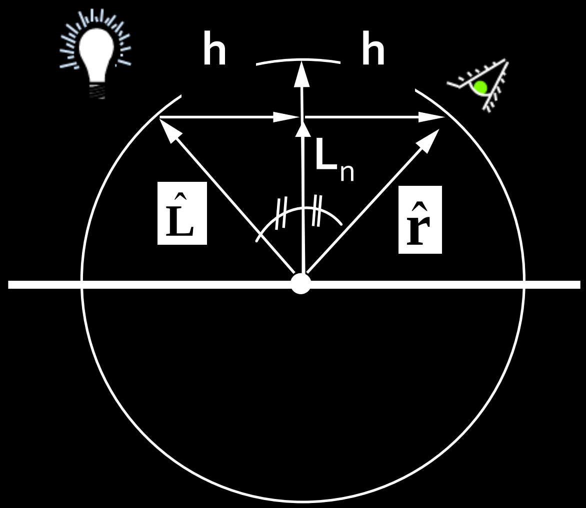
1. $C_{dif_\lambda} = C_{light_\lambda} \, k_{dif_\lambda} (\hat{\mathbf{n}} \cdot \hat{\mathbf{L}})$
2. Light scattered uniformly in all directions: k_{dif_λ}
3. Diffuse Intensity: linear variation with angle cos

Local Specular Reflection Component



$$C_{s\lambda} = C_{luz_\lambda} k_{s\lambda} (\hat{\mathbf{r}} \cdot \hat{\mathbf{v}})^n$$

Mirror Reflection Vector



$$\mathbf{L}_n = (\hat{\mathbf{L}} \cdot \hat{\mathbf{n}})\hat{\mathbf{n}}$$

$$\mathbf{h} = \mathbf{L}_n - \mathbf{L}$$

$$\hat{\mathbf{r}} = \mathbf{L}_n + \mathbf{h}$$

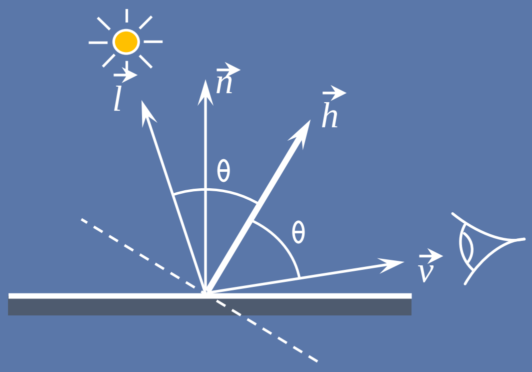
$$\hat{\mathbf{r}} = 2\left(\hat{\mathbf{L}} \cdot \hat{\mathbf{n}}\right)\hat{\mathbf{n}} - \hat{\mathbf{L}}$$

Blinn Approximation

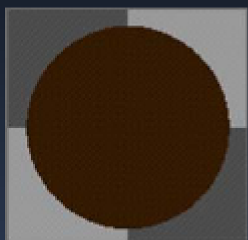
- Computation of $\vec{r}$ is expensive
  - Instead, ***halfway vector*** $\vec{h}$ is used

$$\vec{h} = \frac{\vec{l} + \vec{v}}{\left|\vec{l} + \vec{v}\right|} = \frac{\vec{l} + \vec{v}}{2}$$

Unit Vectors l and v



Blinn-Phong Reflection Model

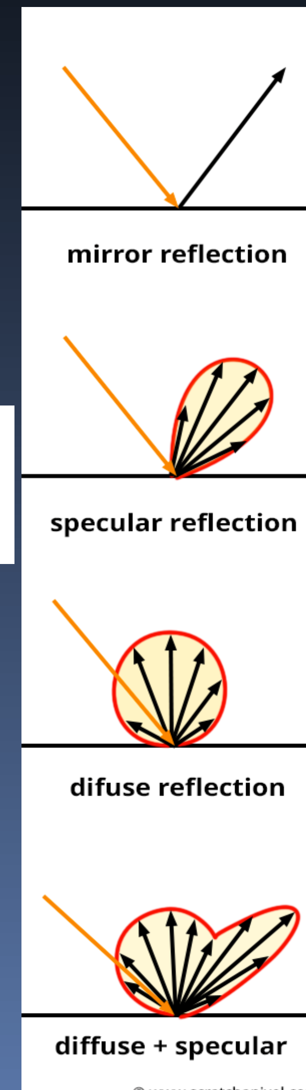


Ambient

Diffuse

Specular

$$C_\lambda = C_{amb_\lambda} + C_{luz_\lambda} k_{dif_\lambda} (\hat{\mathbf{n}} \cdot \hat{\mathbf{L}}) + C_{luz_\lambda} k_{s\lambda} (\hat{\boldsymbol{h}} \cdot \hat{n})^n$$

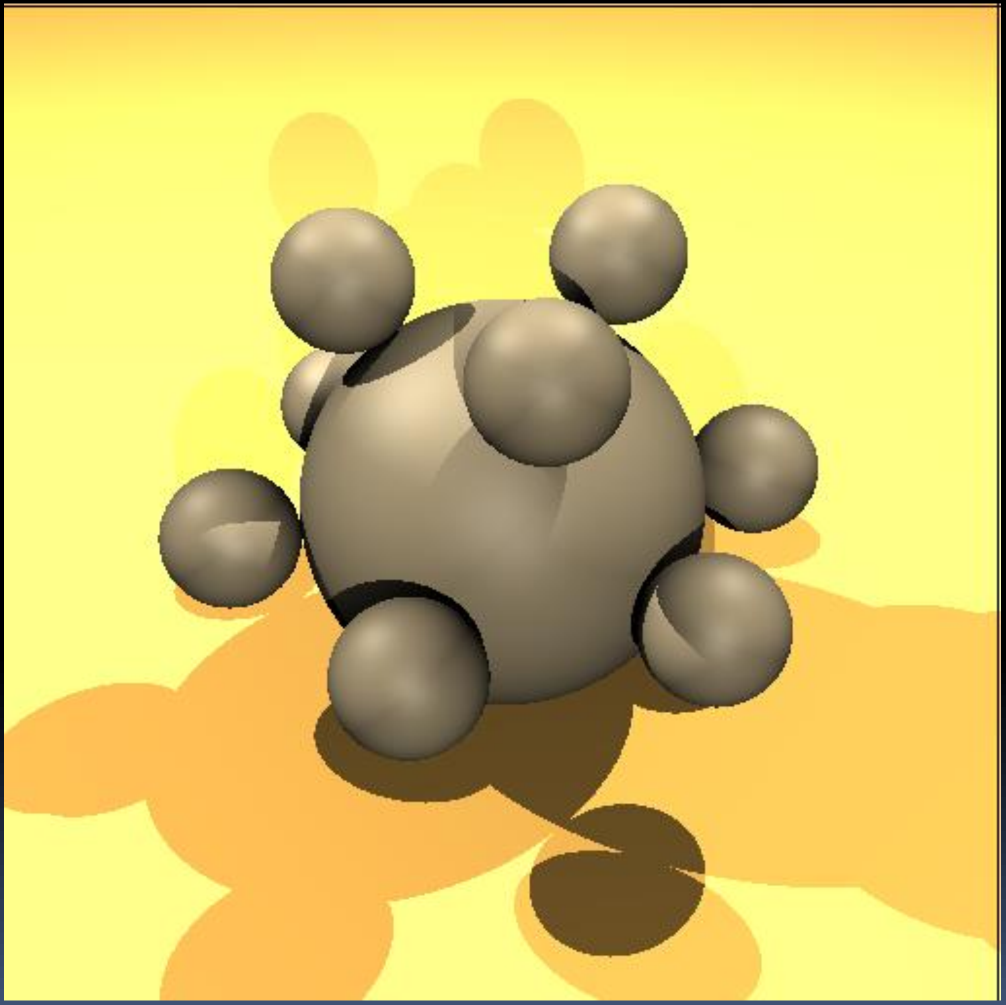


Model with multiple lights and shadow

$$\begin{pmatrix} I_r \\ I_g \\ I_b \end{pmatrix} = \begin{pmatrix} I_{ar} \\ I_{ag} \\ I_{ab} \end{pmatrix} \otimes \begin{pmatrix} k_{dr} \\ k_{dg} \\ k_{db} \end{pmatrix} + \sum_{luzes} f_s \left(\begin{pmatrix} l_r \\ l_g \\ l_b \end{pmatrix} \otimes \begin{pmatrix} k_{dr} \\ k_{dg} \\ k_{db} \end{pmatrix} (\hat{\mathbf{n}} \cdot \hat{\mathbf{L}}) + \begin{pmatrix} l_r \\ l_g \\ l_b \end{pmatrix} \otimes \begin{pmatrix} k_{sr} \\ k_{sg} \\ k_{sb} \end{pmatrix} (\hat{\boldsymbol{h}} \cdot \hat{\mathbf{n}})^n \right)$$

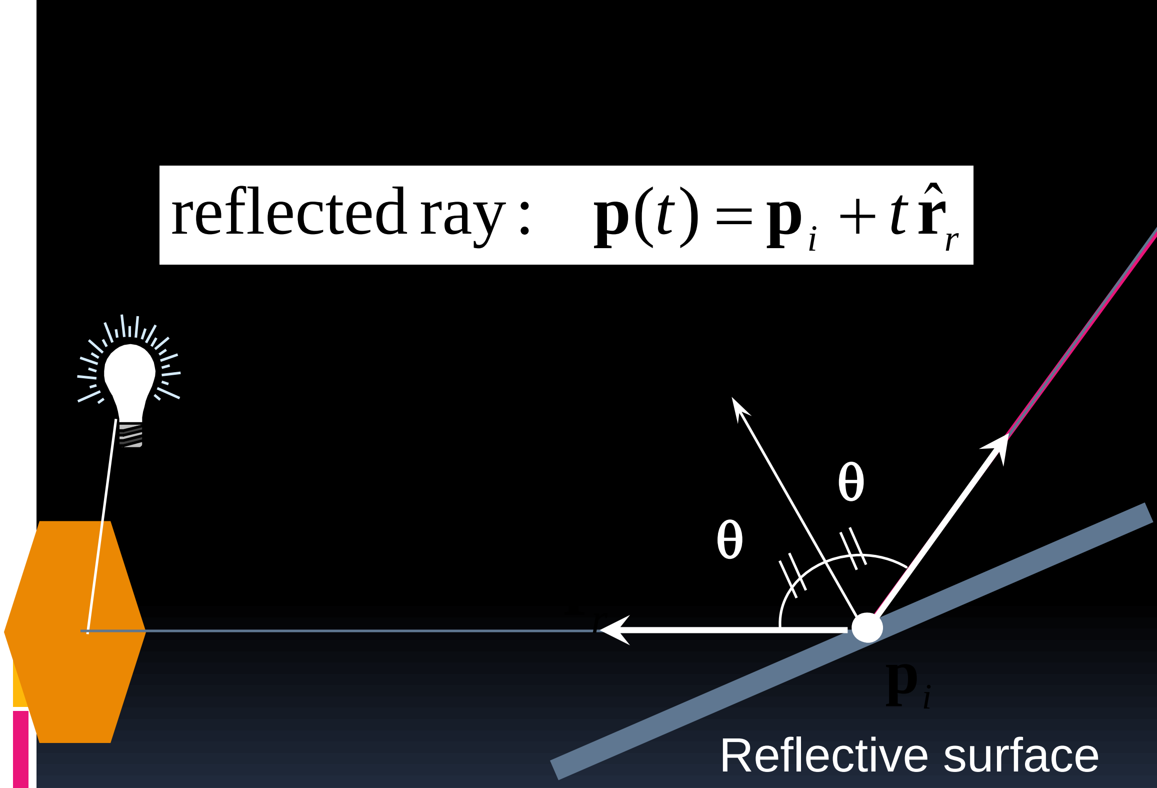
$$f_s = \begin{cases} 0 & \text{if in shadow} \\ 1 & \text{otherwise} \end{cases}$$

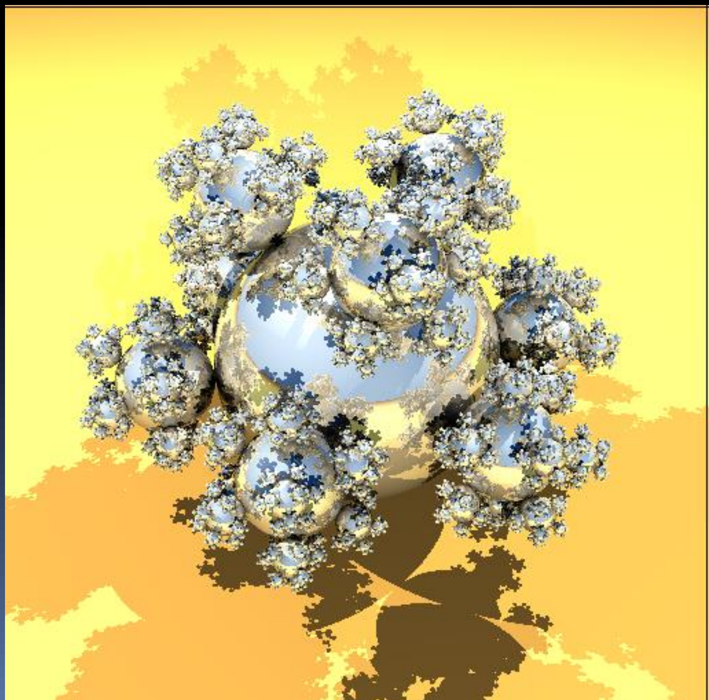
First Lab Exercise



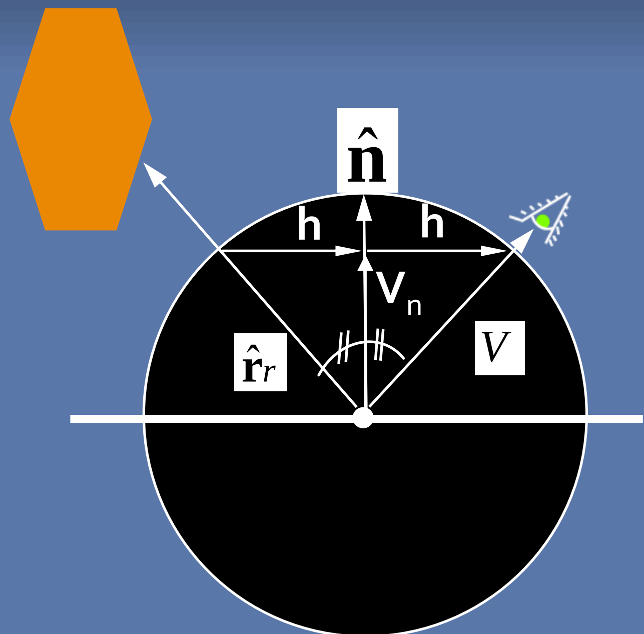
Indirect Illumination: Mirror reflections from other objects

$$\text{reflected ray}: \quad \mathbf{p}(t) = \mathbf{p}_i + t\,\hat{\mathbf{r}}_r$$





Calculating Mirror Reflection Vector



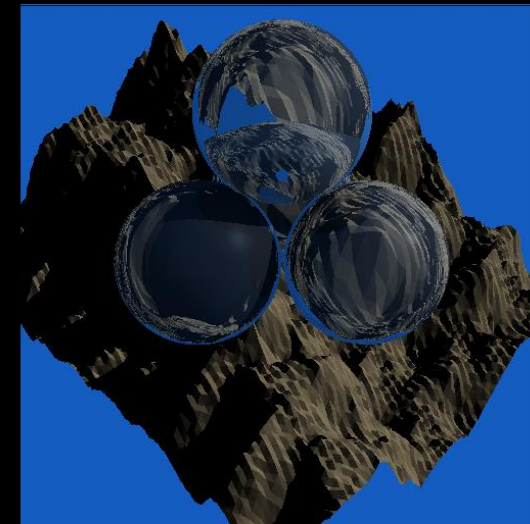
$$\mathbf{h} = V_n - V$$

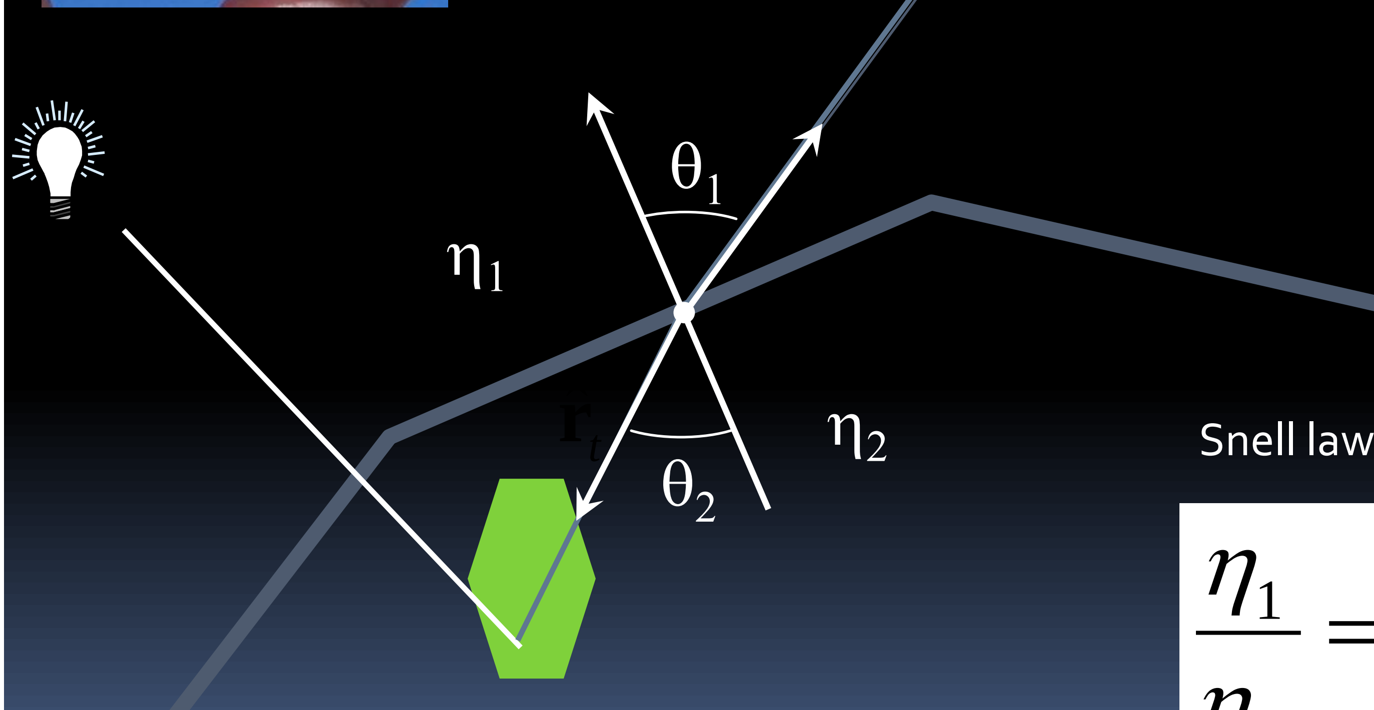
$$V_n = (V \cdot \hat{\mathbf{n}})\hat{\mathbf{n}}$$

$$\hat{\mathbf{r}}_r = V_n + \mathbf{h}$$

$$\hat{\mathbf{r}}_r = 2\left(V \cdot \hat{\mathbf{n}}\right)\hat{\mathbf{n}} - V$$

Transparent objects

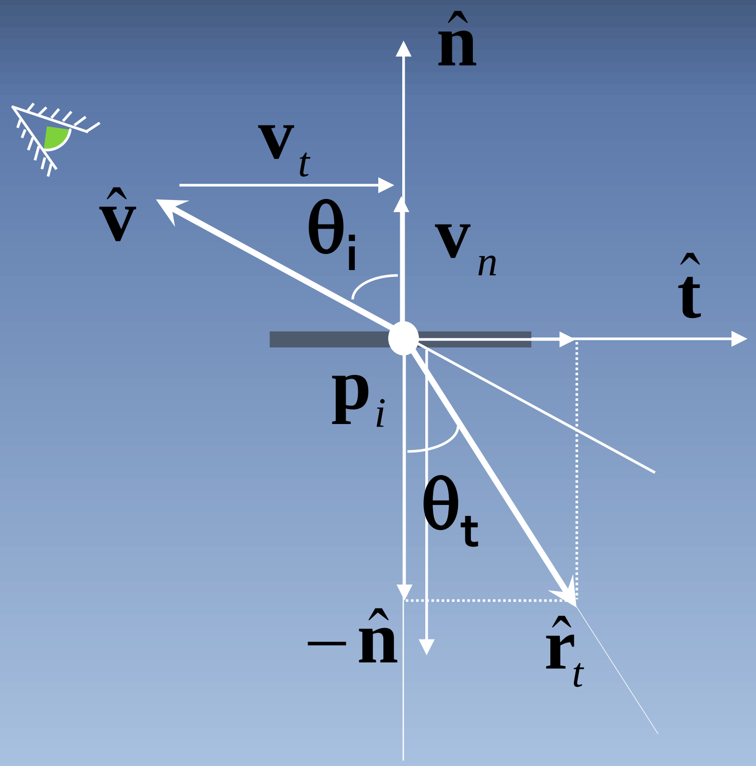




Snell law

$$\frac{\eta_1}{\eta_2} = \frac{\sin \theta_2}{\sin \theta_1}$$

Indirect Illumination: Refracted Ray



$$\mathbf{r}_t = \sin\theta_t \hat{\mathbf{t}} + \cos\theta_t(-\hat{\mathbf{n}})$$

$$\hat{\mathbf{t}} = \frac{1}{\|\mathbf{v}_t\|}\mathbf{v}_t$$

$$\mathbf{v}_t = (\hat{\mathbf{v}} \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} - \hat{\mathbf{v}}$$

$$\|\mathbf{v}_t\| = \sin\theta_i$$

$$\sin\theta_t = \frac{\eta_i}{\eta_t}\sin\theta_i$$

$$\cos\theta_t = \sqrt{1-\sin^2\theta_t}$$

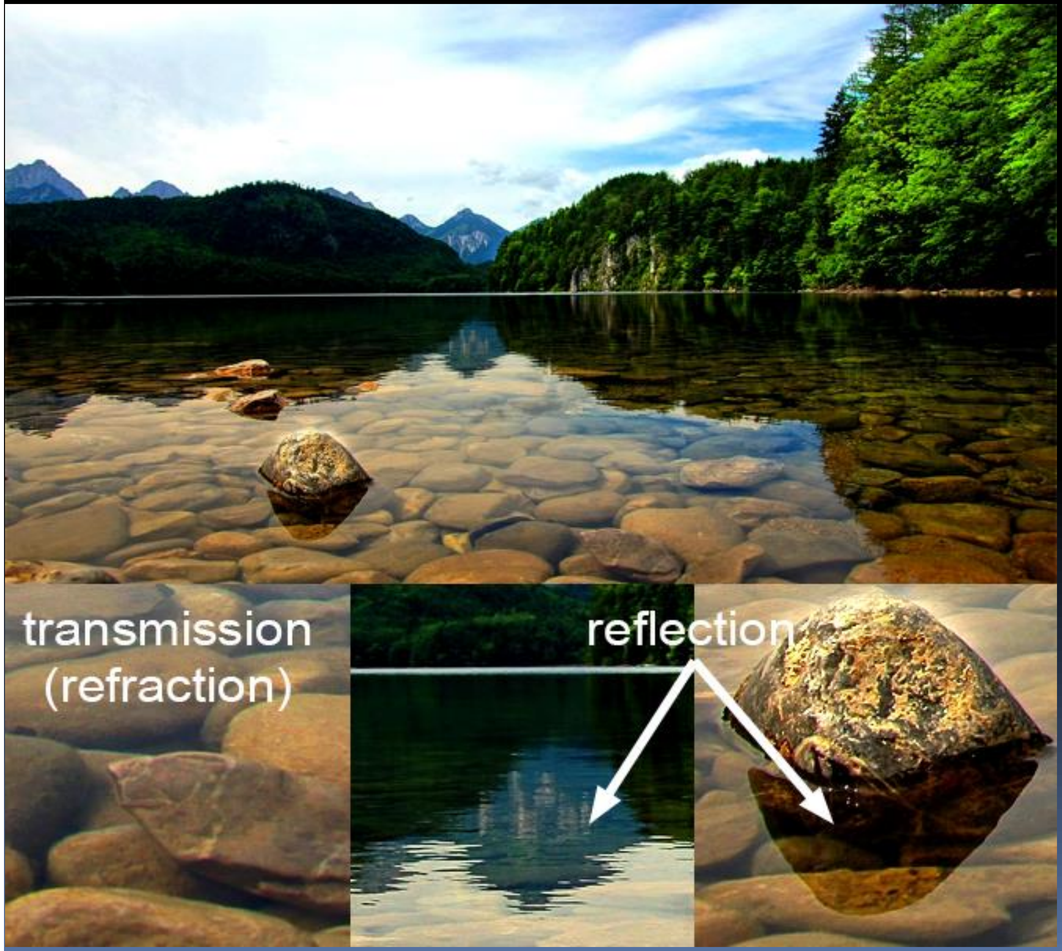
$$\text{refracted ray}: \quad \mathbf{p}(t) = \mathbf{p}_i + t\,\hat{\mathbf{r}}_t$$

Reflective and refractive objects

$$\begin{pmatrix} I_r \\ I_g \\ I_b \end{pmatrix} = \sum_{luzes} f_s \left(\begin{pmatrix} l_r \\ l_g \\ l_b \end{pmatrix} \otimes \begin{pmatrix} k_{dr} \\ k_{dg} \\ k_{db} \end{pmatrix} (\hat{\mathbf{n}} \cdot \hat{\mathbf{L}}) + \begin{pmatrix} l_r \\ l_g \\ l_b \end{pmatrix} \otimes \begin{pmatrix} k_{sr} \\ k_{sg} \\ k_{sb} \end{pmatrix} (\hat{\mathbf{r}}_r \cdot \hat{\mathbf{L}})^n \right) + kr \begin{pmatrix} I_r(\mathbf{r}_r) \\ I_g(\mathbf{r}_r) \\ I_b(\mathbf{r}_r) \end{pmatrix} + (1 - k_r) \begin{pmatrix} I_r(\mathbf{r}_t) \\ I_g(\mathbf{r}_t) \\ I_b(\mathbf{r}_t) \end{pmatrix}$$

Mirror reflection attenuation

Refraction attenuation



Fresnel equations

The amount of reflected vs. refracted light can be computed using the **Fresnel equations**.

Light is composed of two perpendicular waves which we call parallel and perpendicular polarised light.

$$R_s = \left| \frac{n_1 \cos\theta_i - n_2 \cos\theta_t}{n_1 \cos\theta_i + n_2 \cos\theta_t} \right|^2$$

$$R_p = \left| \frac{n_1 \cos\theta_t - n_2 \cos\theta_i}{n_1 \cos\theta_t + n_2 \cos\theta_i} \right|^2$$

$$K_r = \frac{1}{2}(R_s + R_p)$$

$$T = 1 - K_r$$