Global Illumination

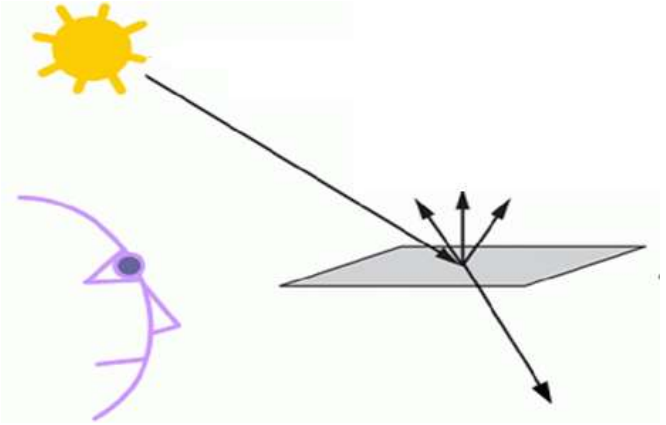
COLLEGE OF COMPUTING

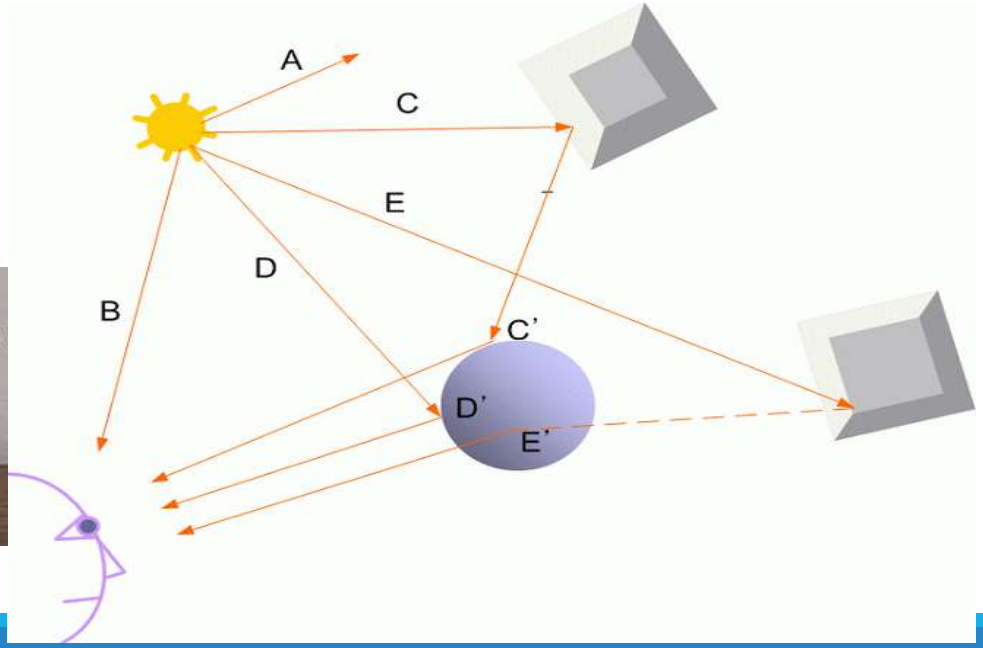
HANYANG ERICA CAMPUS

Q YOUN HONG (홍규연)

Physics on Lights

- 빛은 직진함
- 빛이 물체의 표면에 닿으면
  - Reflection (반사)
  - Refraction (굴절)
  - Absorption (흡수)
- $\mathrm{I}(x, y)$: 이 방향으로 들어오는 모든 빛들의 합



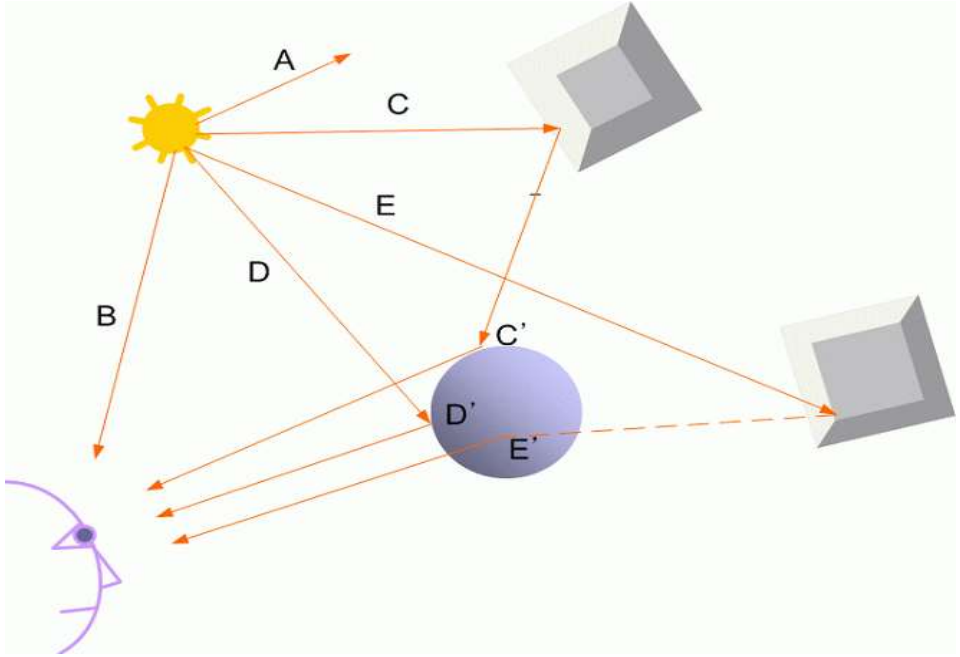


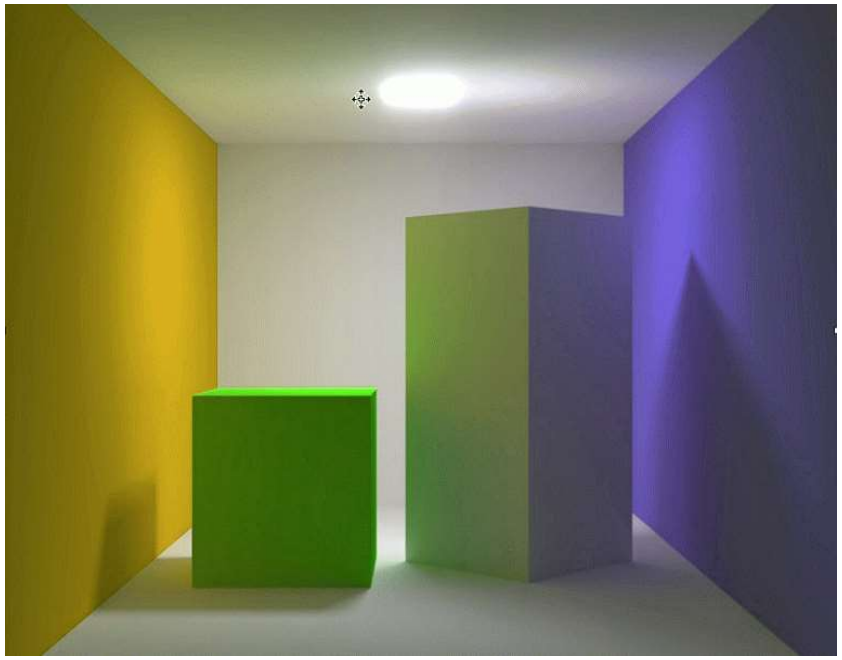
Reflection

Refraction

Global Illumination

Global illumination model: 모든 물체의 표면에서 반사/굴절되는 모든 빛들을 고려하는 모델링





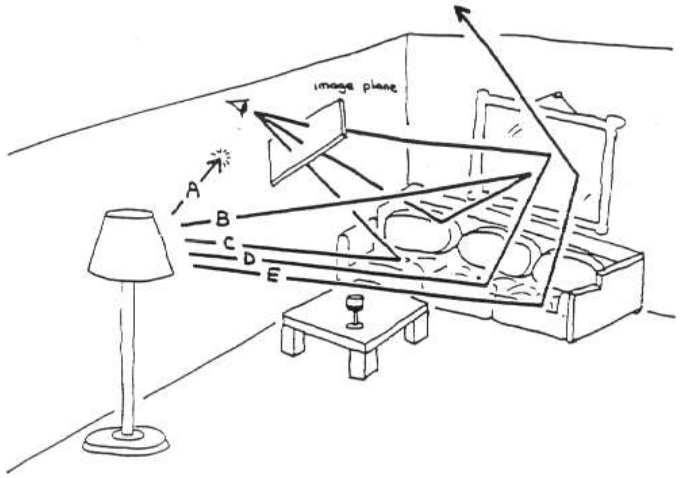
Global Illumination

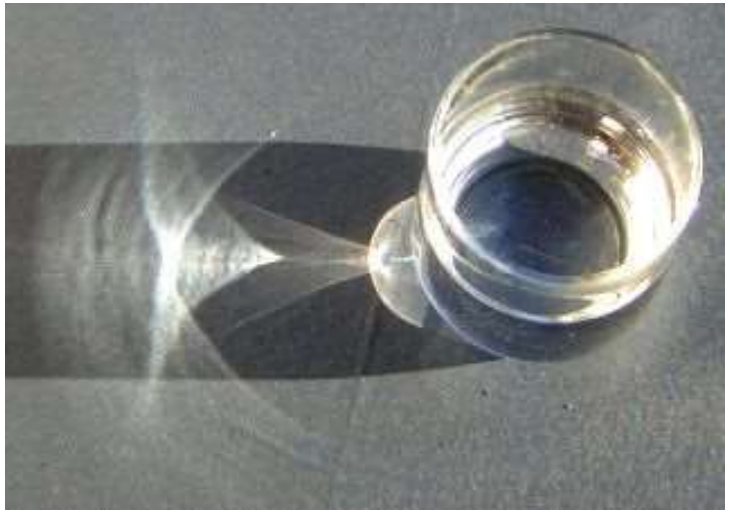
- Local illumination vs. Global illumination
  - Local illumination: 간단함, rasterization 사용, 물체들 간의 interaction을 고려하지 않음
  - Global illumination: 복잡함, 물체들 간의 상호 작용을 고려한 계산 필요

    (computationally intensive algorithms)
- Global illumination을 구현하기 위한 알고리즘

⇒ **Ray-tracing method**

- Reflections
- Refractions and Transparency
- Shadows
- Caustics

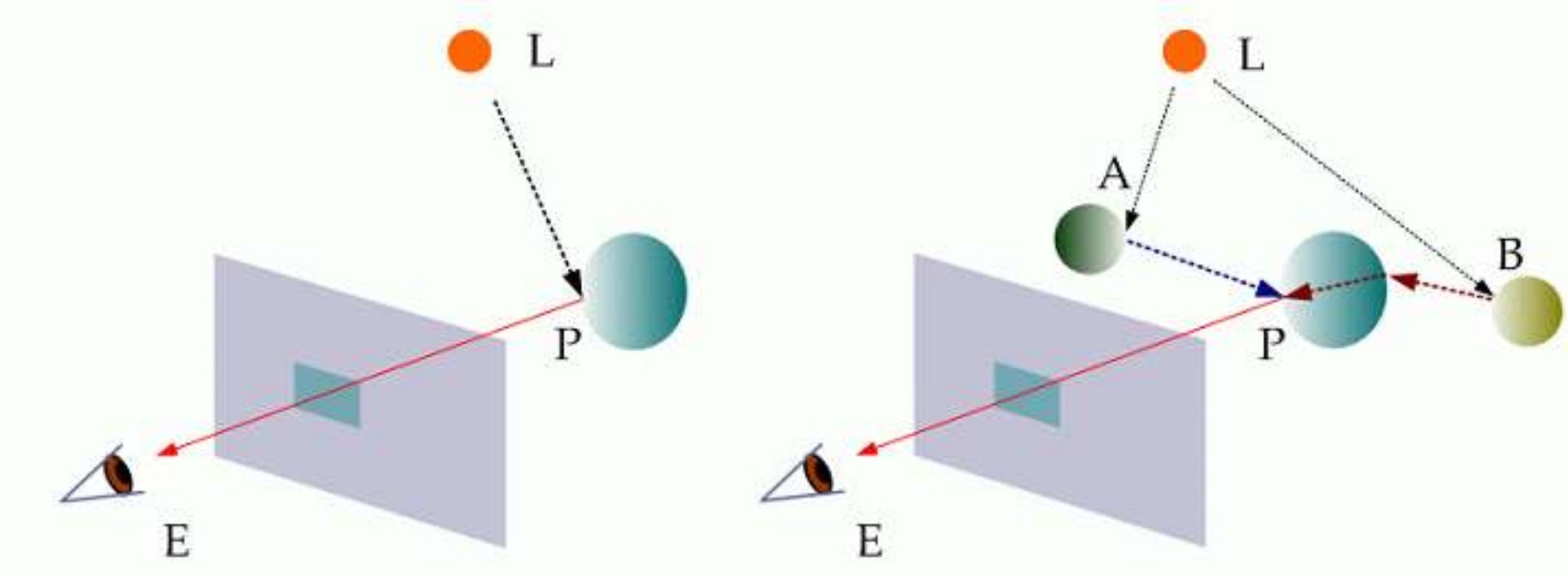




Caustics

Ray-Tracing Algorithm

image-ordered rendering: 이론적으로는, image의 각 pixel을 통해 들어 오는 빛의 양을 계산함



Ray-Tracing Algorithm

Ray-Tracing algorithm

- Viewing ray가 눈에서 시작, image plane의 각 pixel을 통과하여 scene을 trace함

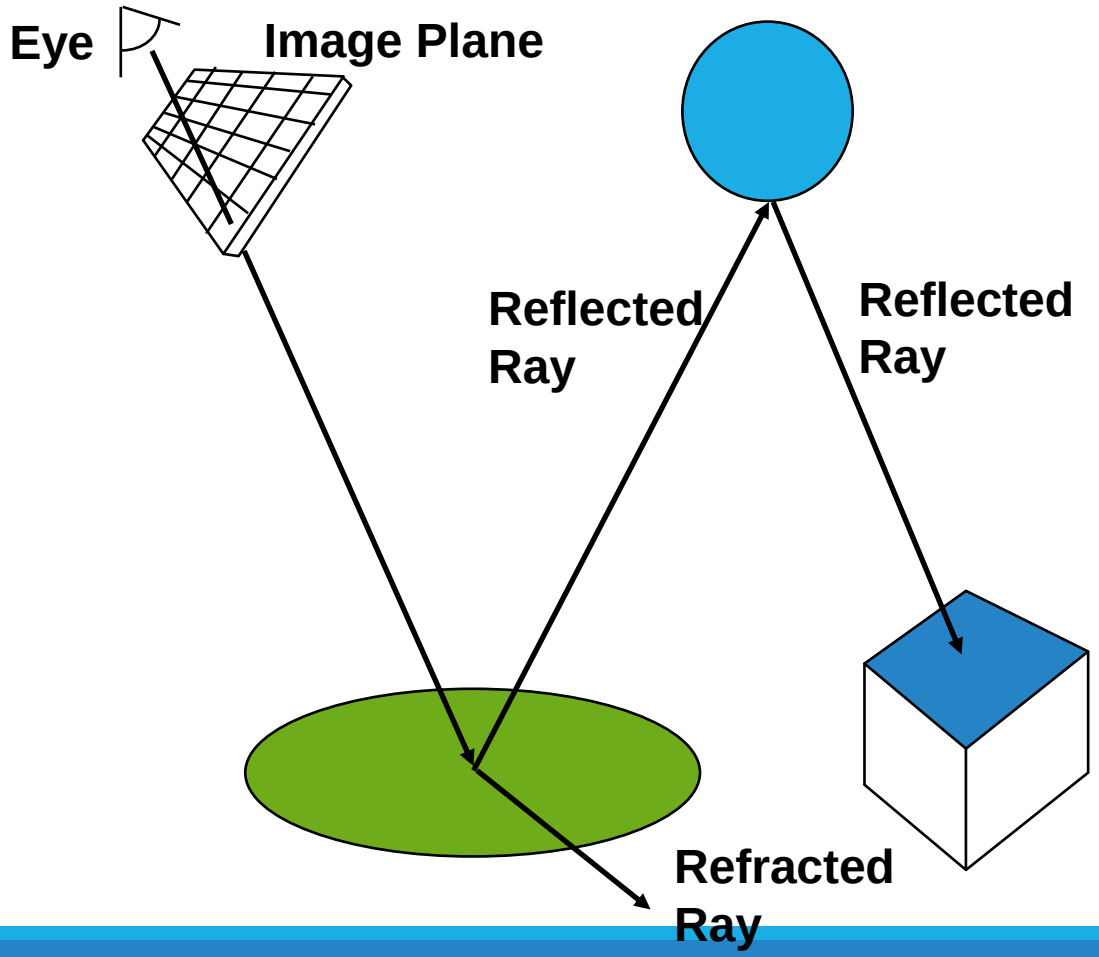
- Ray가 첫 물체에 닿으면, 물체 표면의 성질에 따라 반사/굴절됨

⇒ Secondary ray 발생, 사출

⇒ 다른 물체에 닿으면 다시 반사/굴절

⇒ Recursion continues…

- Ray는 다음의 경우에 멈춤
  - 아무 물체에도 닿지 않을 때
  - 반사/굴절하지 않는 물체에 부딪혔을 때
  - 최대 Recursion depth에 도달했을 때



Basic Ray-Tracing Algorithm (without recursion)


```
for each pixel do
   compute viewing ray;
   find first object hit by ray and its surface normal n;
   set pixel color to value computed from hit point, light, and n;
```

1. Ray generation: 각 pixel의 viewing ray의 시작점과 ray의 방향을 결정
2. Ray intersection: viewing ray와 가장 가깝게 충돌하는 물체(의 표면)을 구하고, 그 위치에서의 법선 벡터 계산
3. Shading: 법선 벡터와 lighting parameter를 이용, shading color 결정

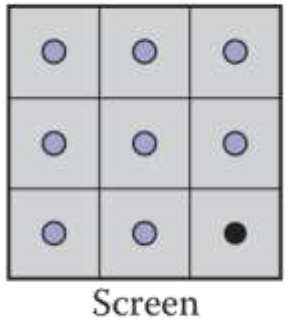
Ray-Tracing: Ray Generation

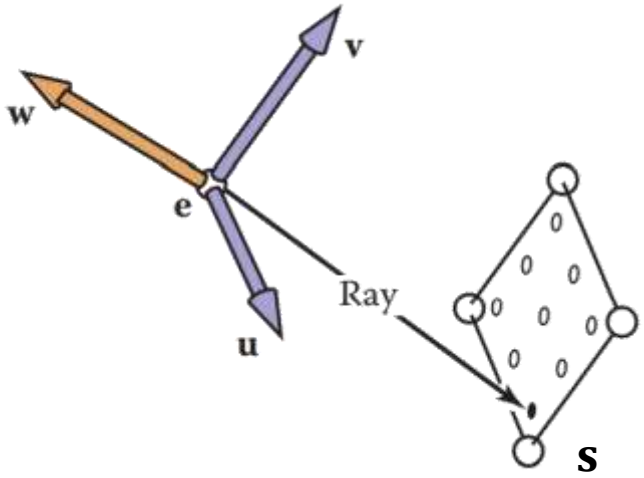
- Viewing ray: eye point $\boldsymbol{e}$, image plane 상의 pixel 위치가 $\boldsymbol{s}$이면,

$$\boldsymbol{p}(t) = \boldsymbol{e} + t(\boldsymbol{s} - \boldsymbol{e}) = \boldsymbol{e} + t\boldsymbol{d}$$

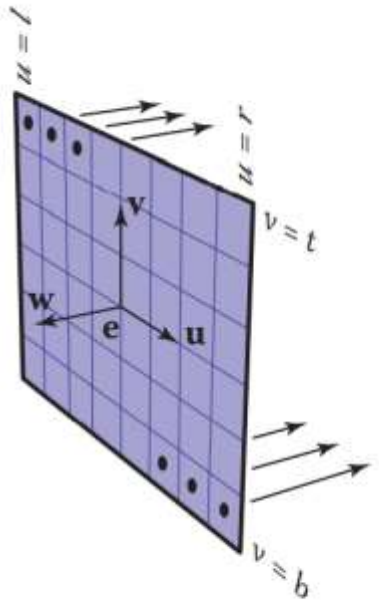
  - $\boldsymbol{p}(0) = \boldsymbol{e}, \quad \boldsymbol{p}(1) = \boldsymbol{s}$
  - $t_1 < t_2$일 때, $\boldsymbol{p}(t_1)$은 $\boldsymbol{p}(t_2)$보다 $\boldsymbol{e}$에 가까이 있음
  - $t < 0$일 때, 눈의 뒤쪽에 위치

- $\boldsymbol{s}$: camera frame에서 정의

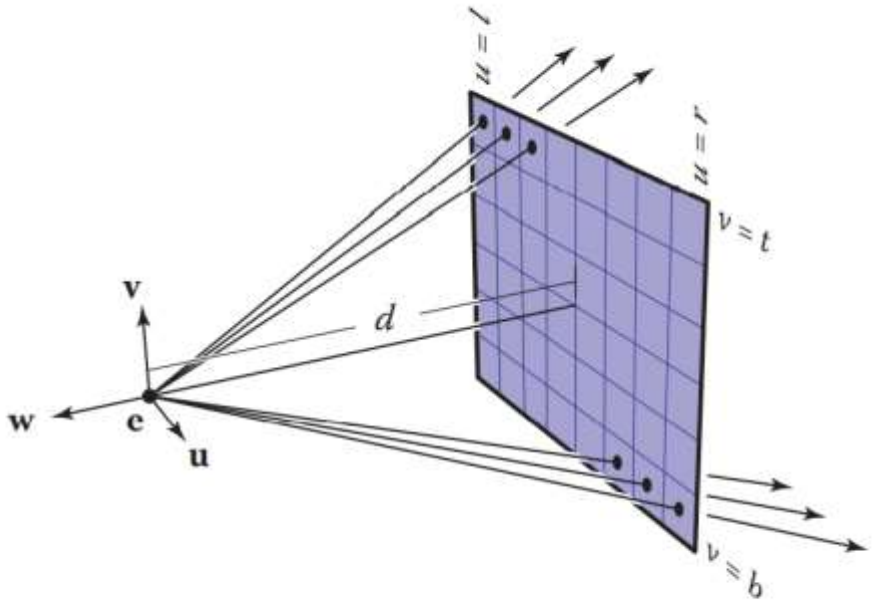




Ray-Tracing: Ray Generation



Parallel projection
same direction, different origins



Perspective projection
same origin, different directions

Ray direction:= $-\boldsymbol{w}$
Ray origin:= $\boldsymbol{e} + u\boldsymbol{u} + v\boldsymbol{v}$

Ray direction:= $-d\boldsymbol{w} + u\boldsymbol{u} + v\boldsymbol{v}$
Ray origin:= $\boldsymbol{e}$

Ray-Object Intersection

- Ray $\boldsymbol{e} + t\boldsymbol{d}$와 제일 먼저 부딪히는 object 계산
  - 제일 작은 양수 t 계산

- Object의 종류에 따라 ray-object intersection algorithm 존재
  - Primitives: box, sphere, cone, cylinder, torus…
  - Implicit surfaces (including quadrics): f(x) = 0
  - Polygons
  - Volumetric data
  - Freeform surfaces: 보통 polygon들의 set으로 근사

Ray-Object Intersection: Ray-Sphere

- 구의 방정식:

$$f(\boldsymbol{x}) = (x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 - R^2 = (\boldsymbol{x} - \boldsymbol{c}) \cdot (\boldsymbol{x} - \boldsymbol{c}) - R^2 = 0$$

- Ray $\boldsymbol{p}(t) = \boldsymbol{e} + t\boldsymbol{d}$와 구가 만나려면,

$f(\boldsymbol{p}(t)) = 0$

$\Rightarrow f(\boldsymbol{e} + t\boldsymbol{d}) = 0$

$\Rightarrow (\boldsymbol{e} + t\boldsymbol{d} - \boldsymbol{c}) \cdot (\boldsymbol{e} + t\boldsymbol{d} - \boldsymbol{c}) - R^2 = 0$

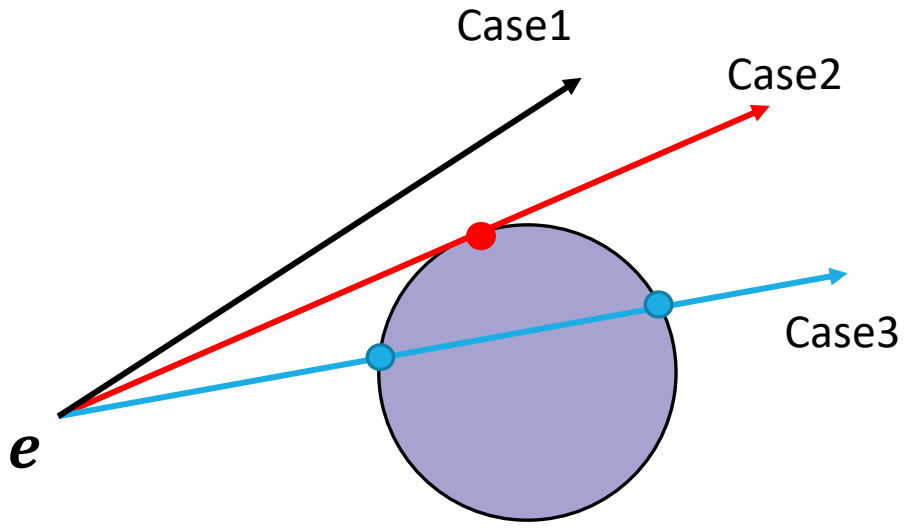
$\Rightarrow \underbrace{(\boldsymbol{d} \cdot \boldsymbol{d})}_{A} t^2 + \underbrace{2\boldsymbol{d} \cdot (\boldsymbol{e} - \boldsymbol{c})}_{B} t + \underbrace{(\boldsymbol{e} - \boldsymbol{c}) \cdot (\boldsymbol{e} - \boldsymbol{c}) - R^2}_{C} = 0$

Case 1. $B^2 - 4AC < 0$: 식의 해 없음 => Ray가 구와 만나지 않음

Case 2. $B^2 - 4AC = 0$: Ray와 구가 한 점에서 만남 ($t = -\frac{B}{2A}$)

Case 3. $B^2 - 4AC > 0$: Ray와 구가 두 점에서 만남($t = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$)

- Ray가 구와 만날 때의 normal $\boldsymbol{n} = \frac{\boldsymbol{p} - \boldsymbol{c}}{R}$



Ray-Object Intersection: Ray-Triangle

- 세 점 $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$로 이루어진 삼각형의 내부의 점 $\boldsymbol{f}(u, v)$:

$$\boldsymbol{f}(u,v) = \boldsymbol{a} + \beta(\boldsymbol{b} - \boldsymbol{a}) + \gamma(\boldsymbol{c} - \boldsymbol{a}), \qquad \beta > 0, \gamma > 0, \beta + \gamma < 1$$

- Ray $\boldsymbol{p}(t) = \boldsymbol{e} + t\boldsymbol{d}$가 삼각형의 내부를 지나기 위해서는

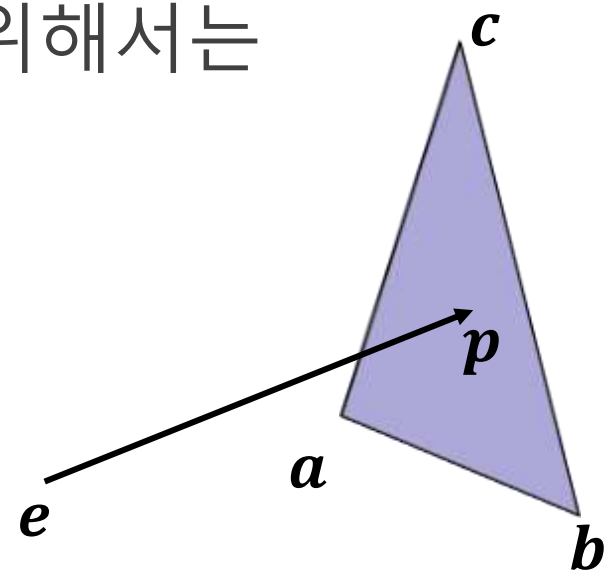
$$\boldsymbol{e} + t\boldsymbol{d} = \boldsymbol{f}(u,v)$$

$$\Rightarrow x_e + t x_d = x_a + \beta(x_b - x_a) + \gamma(x_c - x_a)$$

$$y_e + t y_d = y_a + \beta(y_b - y_a) + \gamma(y_c - y_a)$$

$$z_e + t z_d = z_a + \beta(z_b - z_a) + \gamma(z_c - z_a)$$

의 선형 시스템의 해 (β, γ)가 위의 조건을 만족



Ray-Object Intersection: Ray-Polygon

- Polygon: $\boldsymbol{p}_1, \ldots \boldsymbol{p}_m$의 점들로 구성, 한 평면 위에 있음 (법선 $\boldsymbol{n}$)
- Step 1: Ray $\boldsymbol{p}(t) = \boldsymbol{e} + t\boldsymbol{d}$가 polygon을 포함한 평면과 만날 때의 $t = t_p$ 계산

$$\left(\boldsymbol{e} + t_p\boldsymbol{d} - \boldsymbol{p}_1\right) \cdot \boldsymbol{n} = 0 \rightarrow t_p = \frac{\left(\boldsymbol{p}_1 - \boldsymbol{e}\right) \cdot \boldsymbol{n}}{\boldsymbol{d} \cdot \boldsymbol{n}}$$

- Step 2: $\boldsymbol{p}(t_p)$가 polygon의 내부에 있는지 test
  - Ex) 점 $\boldsymbol{p}(t_p)$과 polygon을 xy-plane으로 투영하고 점에서 2D ray를 그림. 이 ray가 polygon과 홀수 번 만나면 점은 내부에 있고, 짝수 번 만나면 점은 외부에 있음

Ray-Object Intersection

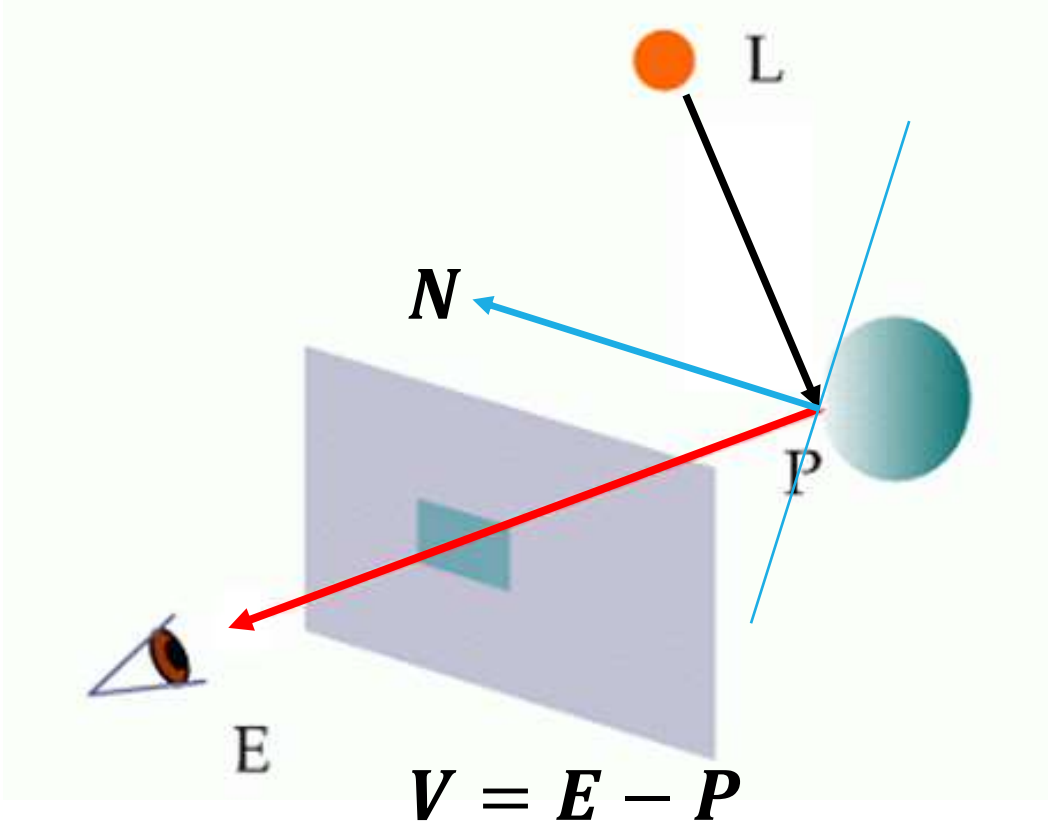
Scene에 여러 물체가 함께 있을 경우: ray와 만나는 물체들 중에서 눈에 가장 가까운 물체만 찾으면 됨

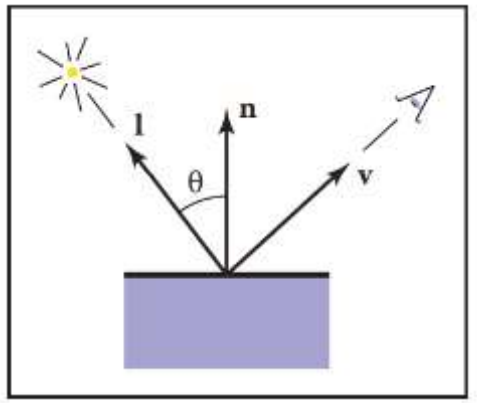
```
// test if ray hits objects in [t0, t1]
hit = false;
for each object o in the group do
   if (o is hit at ray parameter t and t ∈ [t0, t1]) then
      hit = true
      hitobject = o;
      t1 = t;
return hit;
```

Shading

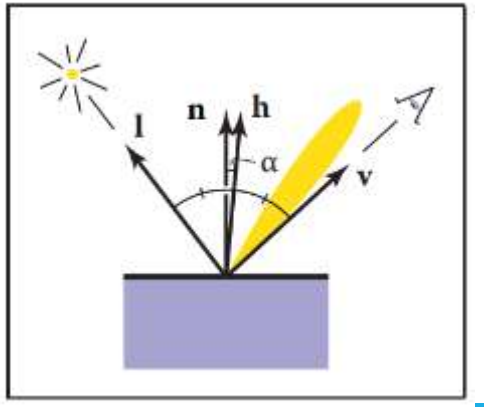
Use shading equation as usual:

$$I = I_a k_a + I_p(k_d\langle N, L\rangle + k_s\langle R, V\rangle^n)$$ ($\langle R,V\rangle$은 $\langle H,N\rangle$으로 대체 가능)





Diffuse light



Specular light

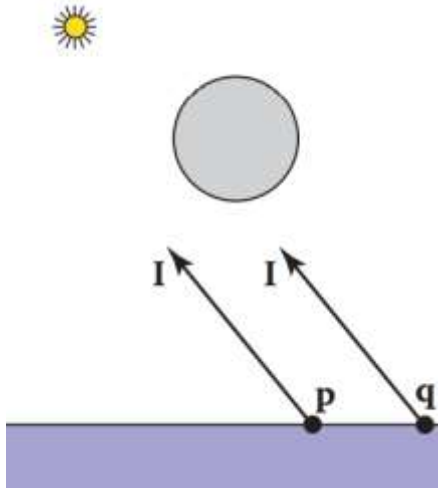
Basic Ray-Tracing Algorithm (without recursion)


```
for each pixel do
   compute viewing ray;
   find first object hit by ray and its surface normal n;
   set pixel color to value computed from hit point, light, and n;
```

```
for each pixel do
   compute viewing ray;
   if (ray hits an object with t ∈ [0,∞)) then
      compute n;
      evaluate shading model and set pixel to that color;
   else
      set pixel color to background color;
```

Shadows

Shadow: 물체 표면 위의 점 p와 광원(light source) 사이에 다른 물체가 존재하면 점 p는 광원의 그림자(shadow)에 속함



⇒광원이 충분히 멀리 떨어져 있을 때,

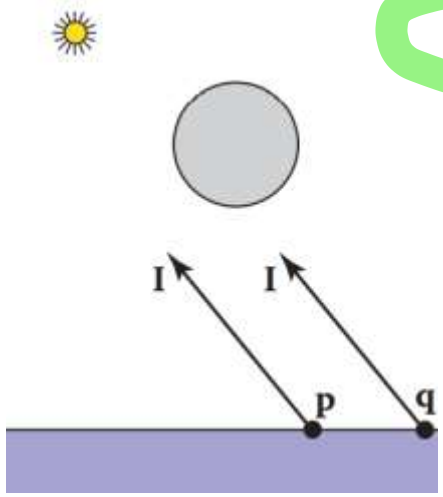
- 점 $\boldsymbol{p}$: $\boldsymbol{p} + t\boldsymbol{l}$은 아무 물체도 닿지 않음 => 광원으로부터 빛을 받음
- 점 $\boldsymbol{q}$: $\boldsymbol{q} + t\boldsymbol{l}$은 구에 충돌 => 광원으로 부터 빛을 받지 못함 => 그림자

⇒Ray $\boldsymbol{l}$: shadow ray

Shadows

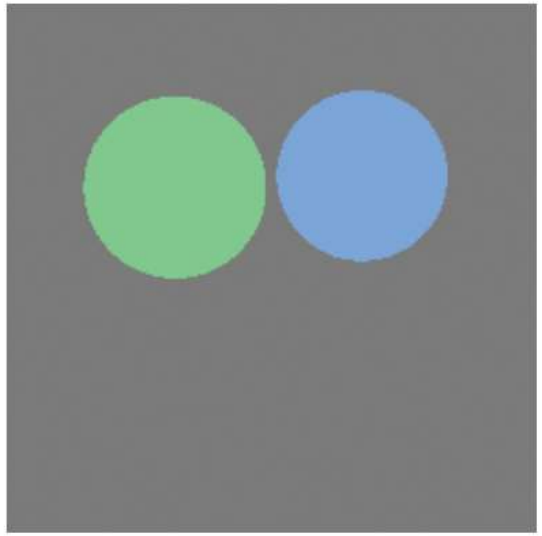
Simulating shadows:

- ray-object intersection에서 구한 점 $\boldsymbol{p}$ 에서 shadow ray $\boldsymbol{l}$를 사출
- $\boldsymbol{p} + t\boldsymbol{l}$이 $t \in [\varepsilon, \infty)$에서 다른 물체에 부딪히면 $\boldsymbol{p}$는 shadow

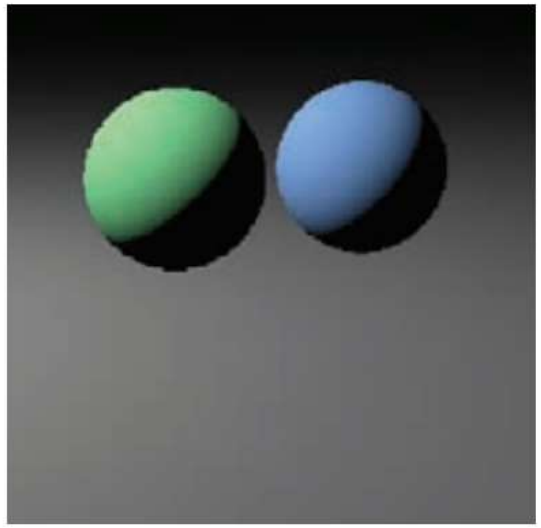



```
function raycolor (ray e + td, real t0, real t1)
  hit-record rec, srec;
  if (scene->hit(e + td, t0, t1, rec)) then
    p = e + (rec.t)d;
    color c = rec.kaIa;
    if (not scene->hit(p + sl, ε, ∞)) then
      vec3 h = normalized(normalized(l) + normalized(−d));
      c = c + rec.kdI max(0, rec.n · l) + rec.ksI (rec.n · h)^rec.n;
      return c;
    else
      return background-color;
```

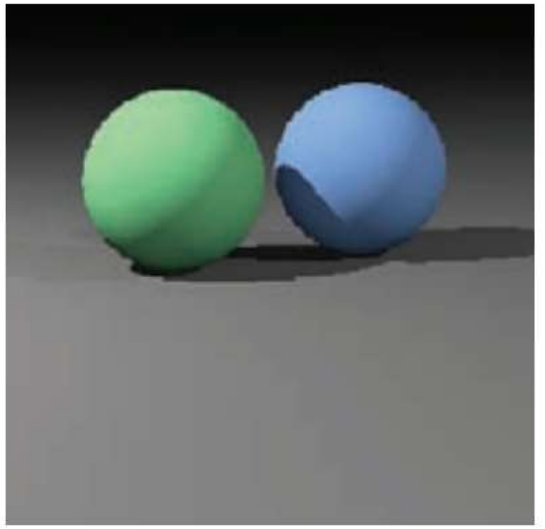
Shadows



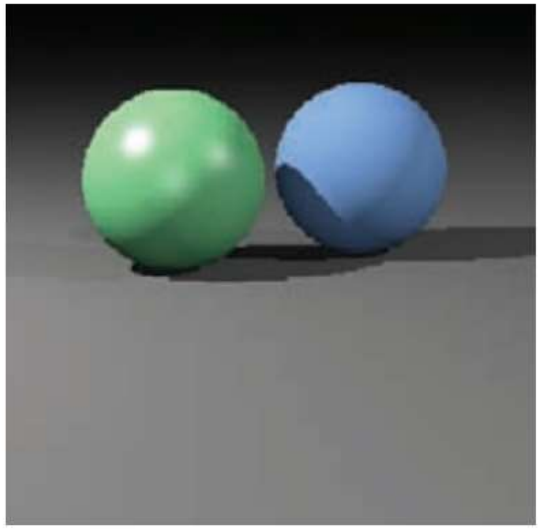
Ray generation + ray-object intersection only (fixed color for each object)



Diffuse shading + a single light source



Diffuse shading + three light sources + shadows

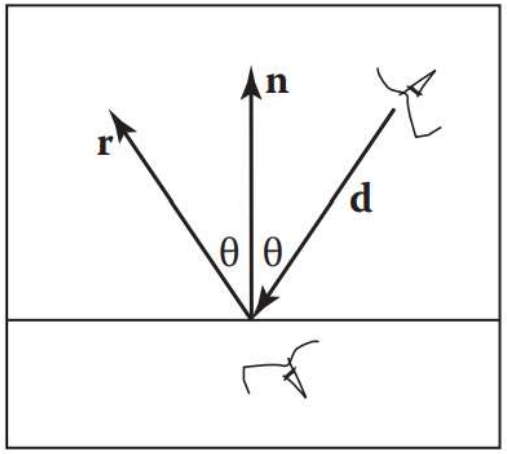


Diffuse shading + specular shading + three light sources + shadows

Ideal Specular Reflection

Add mirror reflection to a ray-tracer

- Viewer는 $\boldsymbol{e}$에서 $\boldsymbol{d}$가 반사된 $\boldsymbol{r}$도 봄
- $\boldsymbol{r} = \boldsymbol{d} - 2(\boldsymbol{d} \cdot \boldsymbol{n})\boldsymbol{n}$
- Color $c = c + k_m \, \text{raycolor}(\boldsymbol{p} + s\boldsymbol{r}, \epsilon, \infty)$



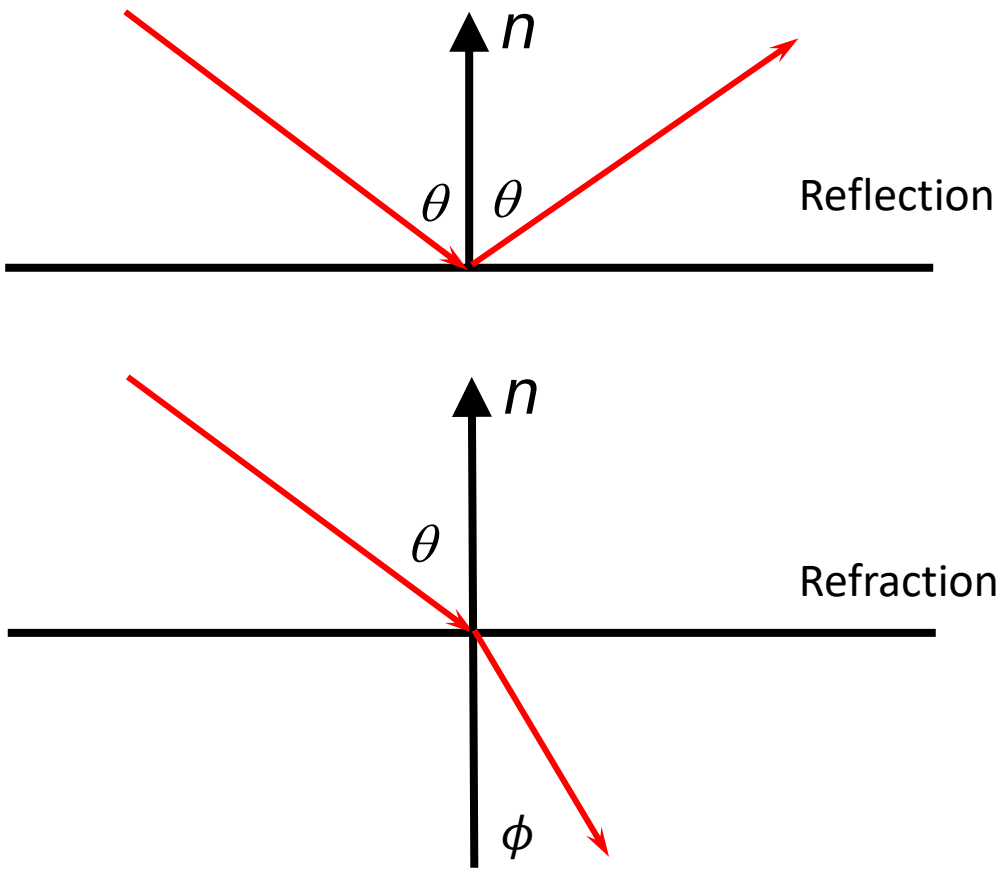
Reflection and Refraction

- Dielectric material (유전체)
  - 다이아몬드, 유리, 물, 공기 등
  - 빛을 굴절시키는 투명한 물질들

- Snell's law – 빛이 굴절률 (refractive index)이 n인 매개체 (medium)에서 굴절률이 n_t인 매개체로 이동할 때,

$$n \sin\theta = n_t \sin\phi$$

$$\Rightarrow \cos^2\phi = 1 - \frac{n^2(1-\cos^2\theta)}{n_t^2}$$



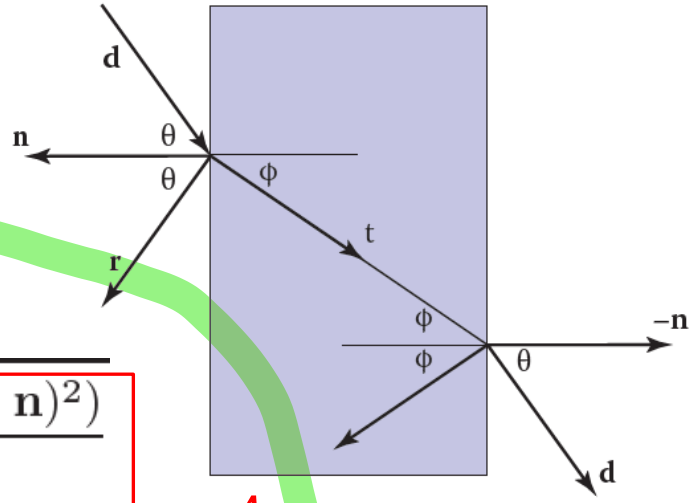
Reflection and Refraction

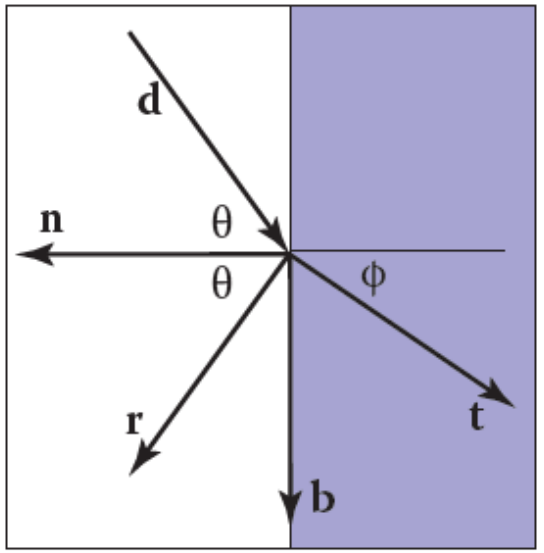
- Refracted ray $\boldsymbol{t}$:

$$\boldsymbol{t} = \sin\phi \boldsymbol{b} - \cos\phi \boldsymbol{n}$$

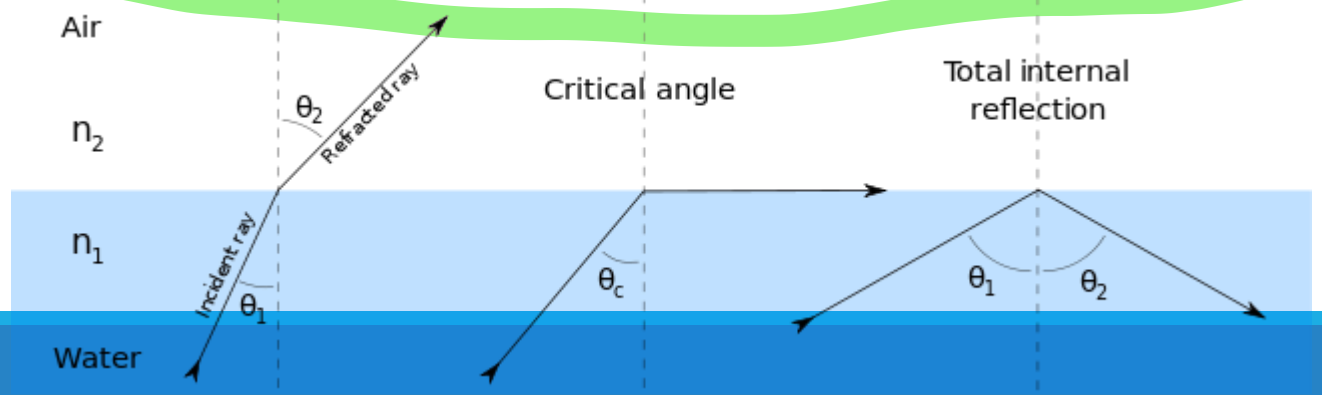
$$\boldsymbol{d} = \sin\theta \boldsymbol{b} - \cos\theta \boldsymbol{n}$$

$$\overrightarrow{\boldsymbol{t}} = \frac{n\,(\mathbf{d} - \mathbf{n}(\mathbf{d}\cdot\mathbf{n}))}{n_t} - \mathbf{n}\sqrt{1 - \frac{n^2\,(1-(\mathbf{d}\cdot\mathbf{n})^2)}{n_t^2}} = A$$



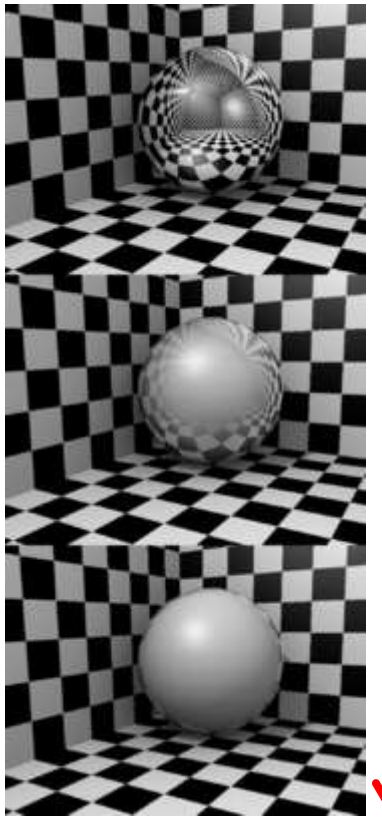


- If A < 0: 굴절되는 빛 없이 들어온 빛이 모두 반사 (total internal reflection)



Fresnel Equation

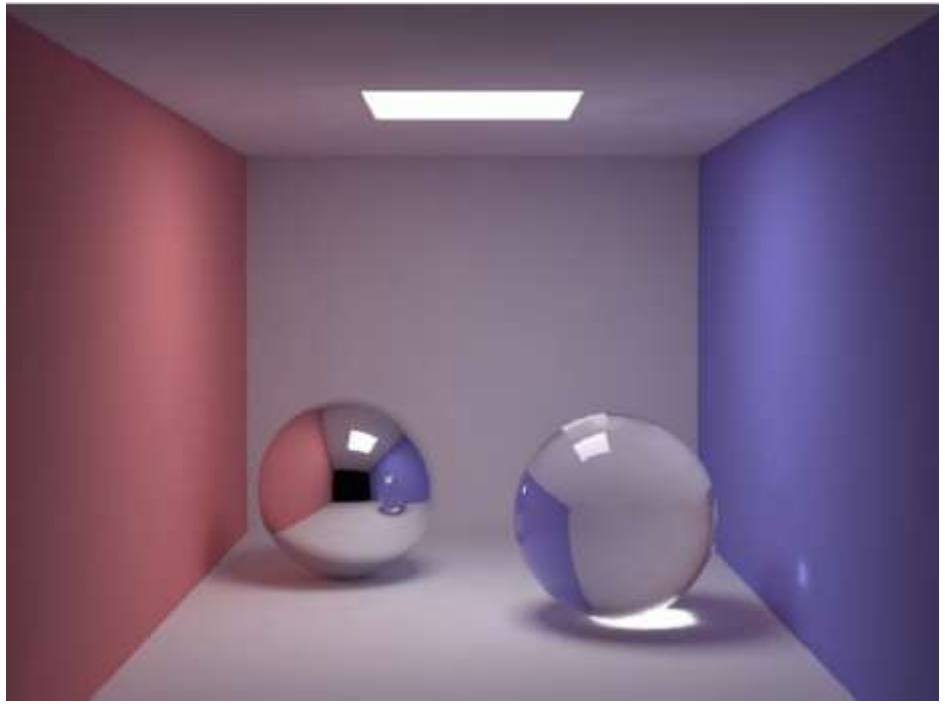
- Fresnel equation은 빛이 다른 매개에서 반사 효과가 얼마나 달라지는지를 설명함
  - $fr = offset + scale\,(1 + V \cdot N)^{n}$
    - $offset$: 기본값 ≈ 0
    - $scale$: 가중치 ≈ 0.1~0.5
  - Coefficient n ≈ 0.5~10
    - n이 클수록 반사의 정도가 감소함
  - 반사의 강도는 또한 입사각에 따라 달라짐



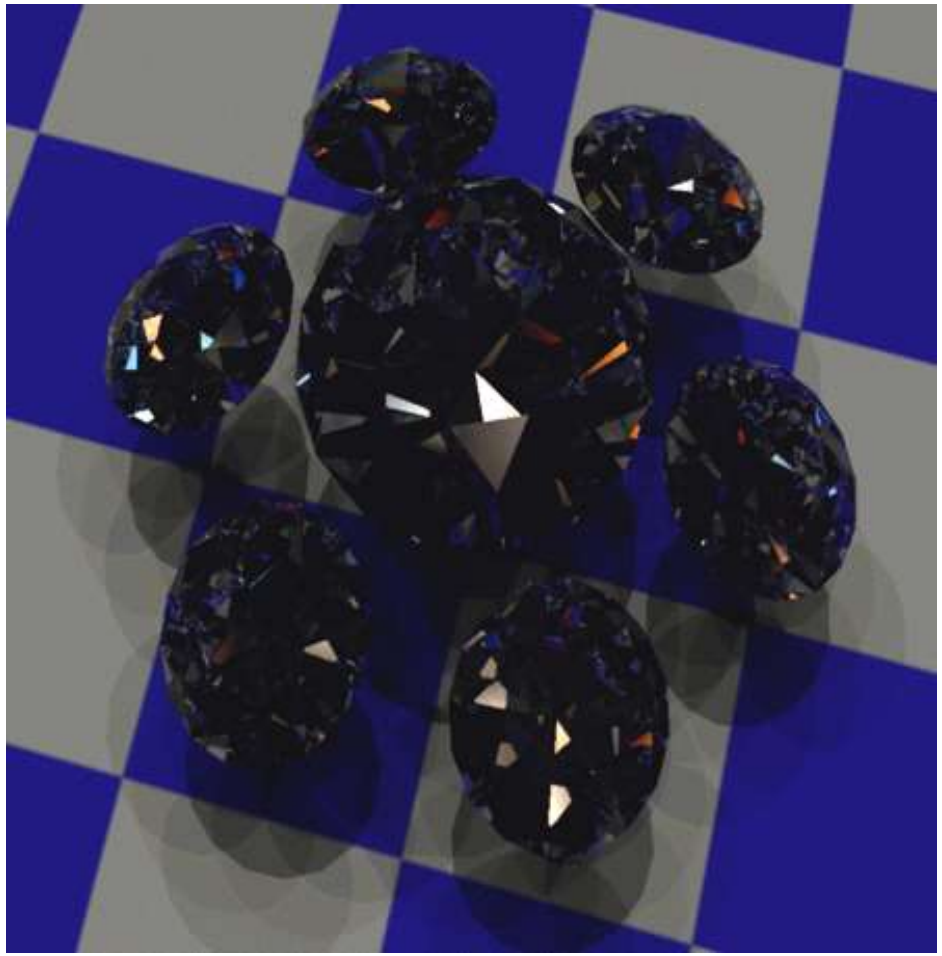
Ray-Tracing (with Reflection and Refraction)


```
function rayTrace(ray e + td, real t0, real t1, scene)
  hit-record rec;
  if (not scene->hit(e + td, t0, t1, rec) then return background-color;
  else
    if ( reflect(rec.obj) ) then
       reflectColor := rayTrace(reflectRay(ray e + td, rec.obj), ϵ, ∞,scene);
    else
       reflectColor := Black;
    if ( dielectric(rec.obj) ) then
       refractColor := rayTrace(refractRay(ray e + td, rec.obj), ϵ, ∞,scene);
    else
       refractColor := Black;
   return Shade(reflectColor, refractColor, obj);
```

Ray-Traced Scenes



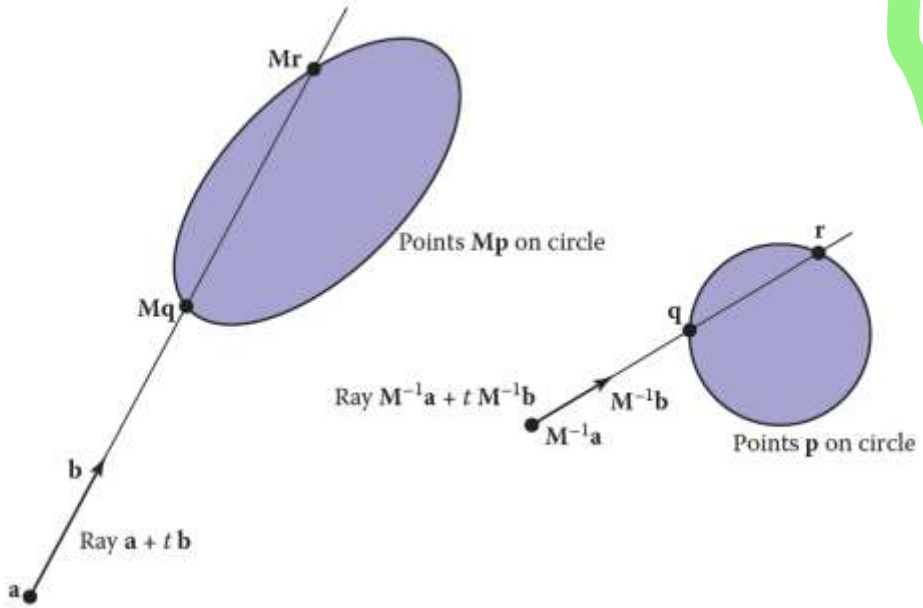
Ray-Traced Scenes



Instancing

Transformation matrix M이 적용된 object $\boldsymbol{o}$와 ray $\boldsymbol{r}$의 ray-object intersection

$\Rightarrow$ $\boldsymbol{o}$와 $M^{-1}\boldsymbol{r}$의 ray-object intersection




```
function hit(Ray e + td, real t0, real t1,
hit-record rec)
 Ray r' = M^-1 a + M^-1 b;
 if (base-object->hit(r', t0, t1, rec)) then
    rec.n = (M^-1)^T rec.n;
    return true;
  else
    return false;
```

Optimized Ray Tracing

- 기본적인 ray-tracing algorithm은 간단하지만, 매우 계산의 양의 많음
  - 모든 pixel에 대해서, recursion의 depth가 깊어질수록 계산하는 ray의 수가 기하급수적으로 증가
- Optimized ray-tracing
  - Trace하는 ray의 수를 줄이기
  - Ray-object intersection 계산의 수를 줄이기
- 많이 쓰이는 방법들
  - Bounding Boxes
  - Object Hierarchies
  - Spatial Subdivision (Octrees/BSP)
  - Tree Pruning (Randomized)

Advanced Phenomena

- Ray tracer는 다음과 같은 현상을 시뮬레이션할 수 있음
  - Fog
  - Soft Shadows
  - Frequency Dependent Light

  (Snell's law는 다른 파장에 대하여 다르게 적용)
- Ray tracer는 diffuse-diffuse interaction 등의 표현에는 어려움이 있음

⇒ Radiosity 도입