MAC420/5744: Introdução à Computação Gráfica

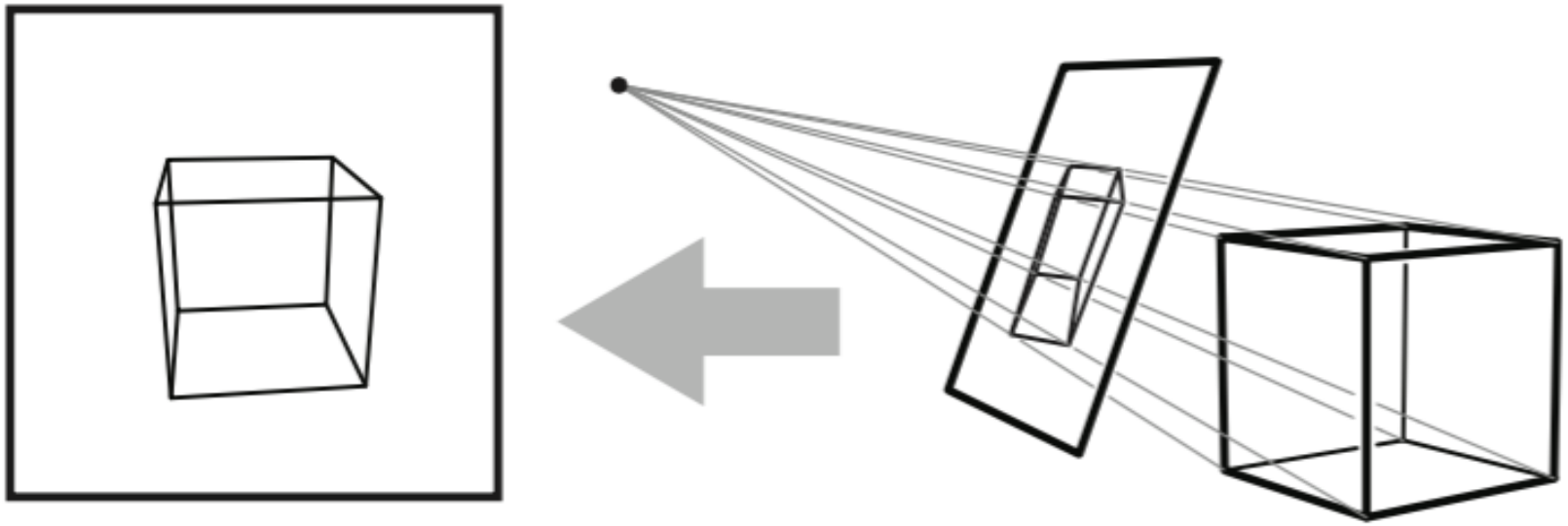
Marcel P. Jackowski
mjack@ime.usp.br

Aula #4: Ray tracing

Some slides by Steve Marschner, Cornell CS4620/5620

Projeção

- To render an image of a 3D scene, we *project* it onto a plane
- Most common projection type is *perspective projection*



Duas abordagens

```
for each object in the scene {
    for each pixel in the image {
        if (object affects pixel) {
            do something
        }
    }
}
```

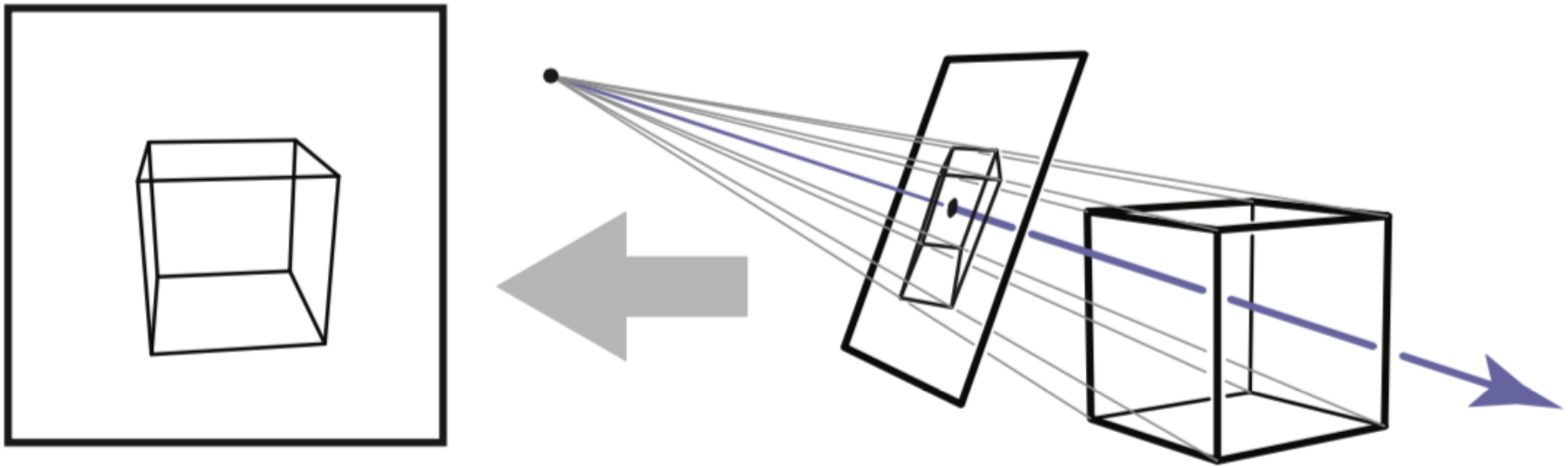
object order
or
rasterization

```
for each pixel in the image {
    for each object in the scene {
        if (object affects pixel) {
            do something
        }
    }
}
```

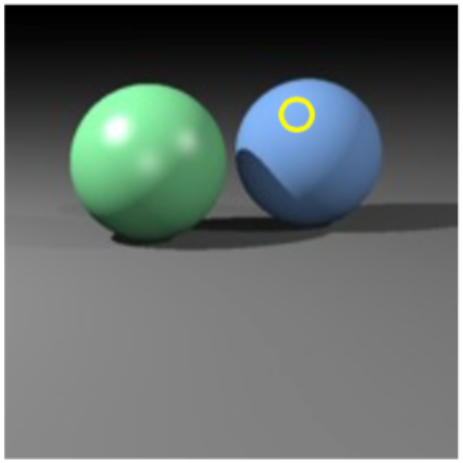
image order
or
ray tracing

Ray tracing

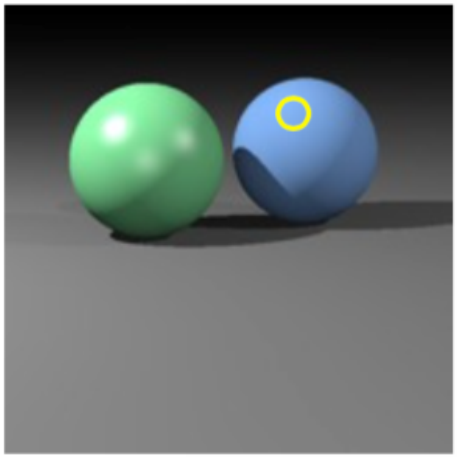
- Start with a pixel – what belongs at that pixel
- Set of points that project to a point in the image



Ray tracing

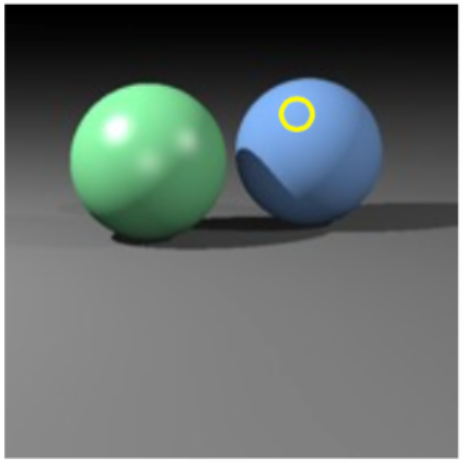


Ray tracing



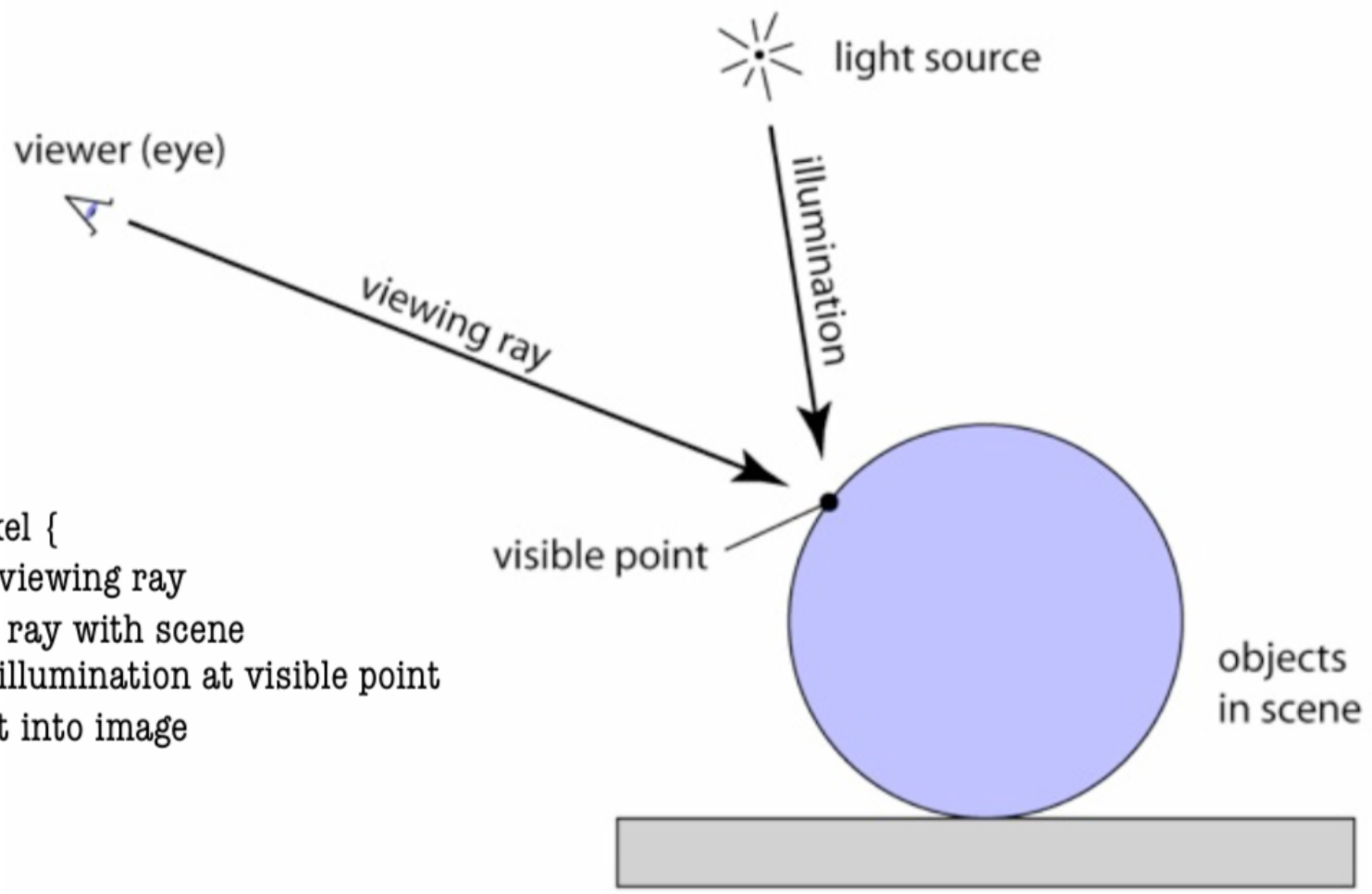
light source
viewer (eye)
viewing ray
visible point
objects
in scene

Ray tracing



light source
viewer (eye)
illumination
viewing ray
visible point
objects
in scene

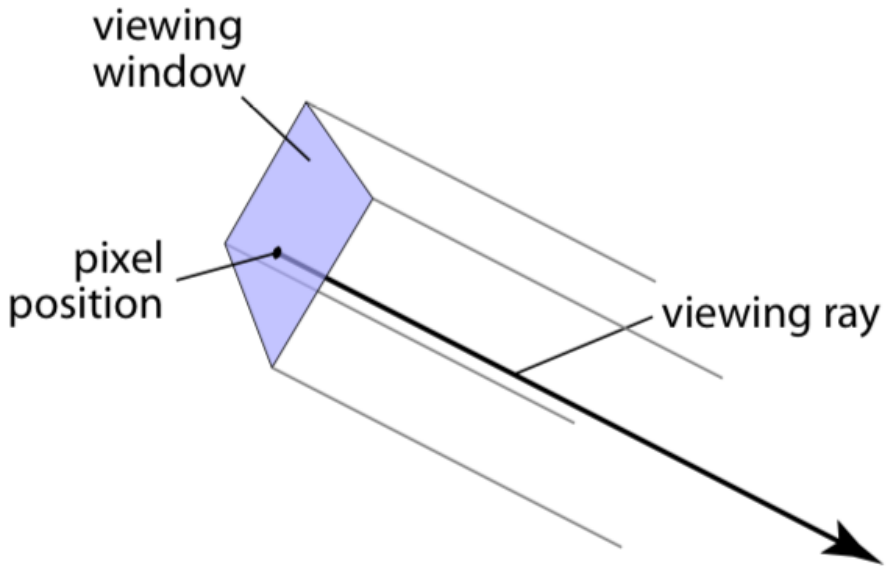
Algoritmo de ray tracing




```
for each pixel {
    compute viewing ray
    intersect ray with scene
    compute illumination at visible point
    put result into image
}
```

Geração de raios - paralela

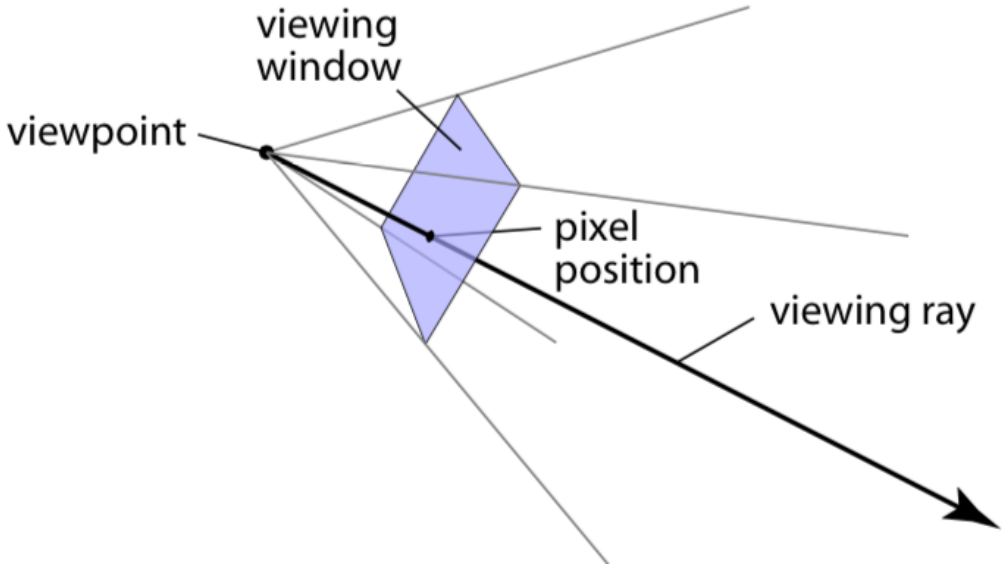
- Ray origin (varying): pixel position on viewing window
- Ray direction (constant): view direction



Planar projection

Geração de raios - perspectiva

- Ray origin (constant): viewpoint
- Ray direction (varying): toward a pixel position on the viewing window



Perspective projection

Interface para as câmeras

- Key operation: generate ray for image position

```
class Camera {
    ...
    Ray generateRay(int col, int row);
}
```

← args go from 0, 0 to width – 1, height – 1

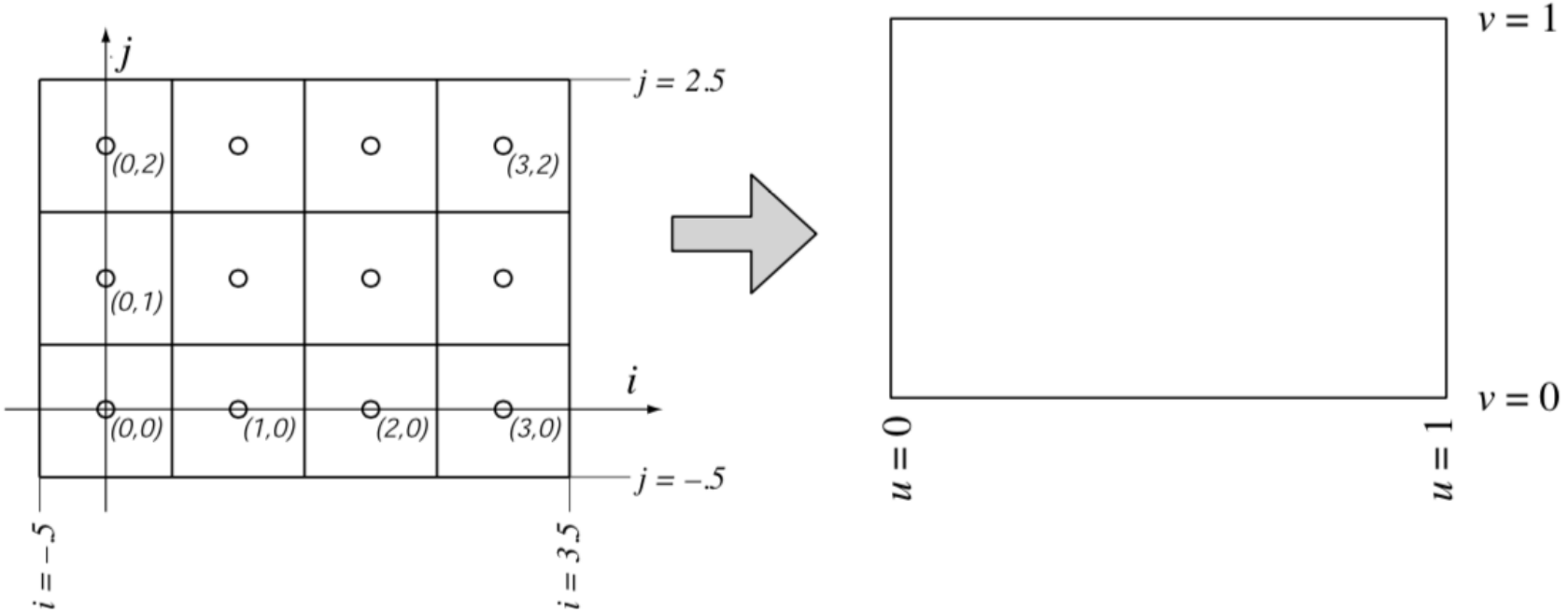
- Modularity problem: Camera shouldn't have to worry about image resolution
  - better solution: normalized coordinates

```
class Camera {
    ...
    Ray generateRay(float u, float v);
}
```

← args go from 0, 0 to 1, 1

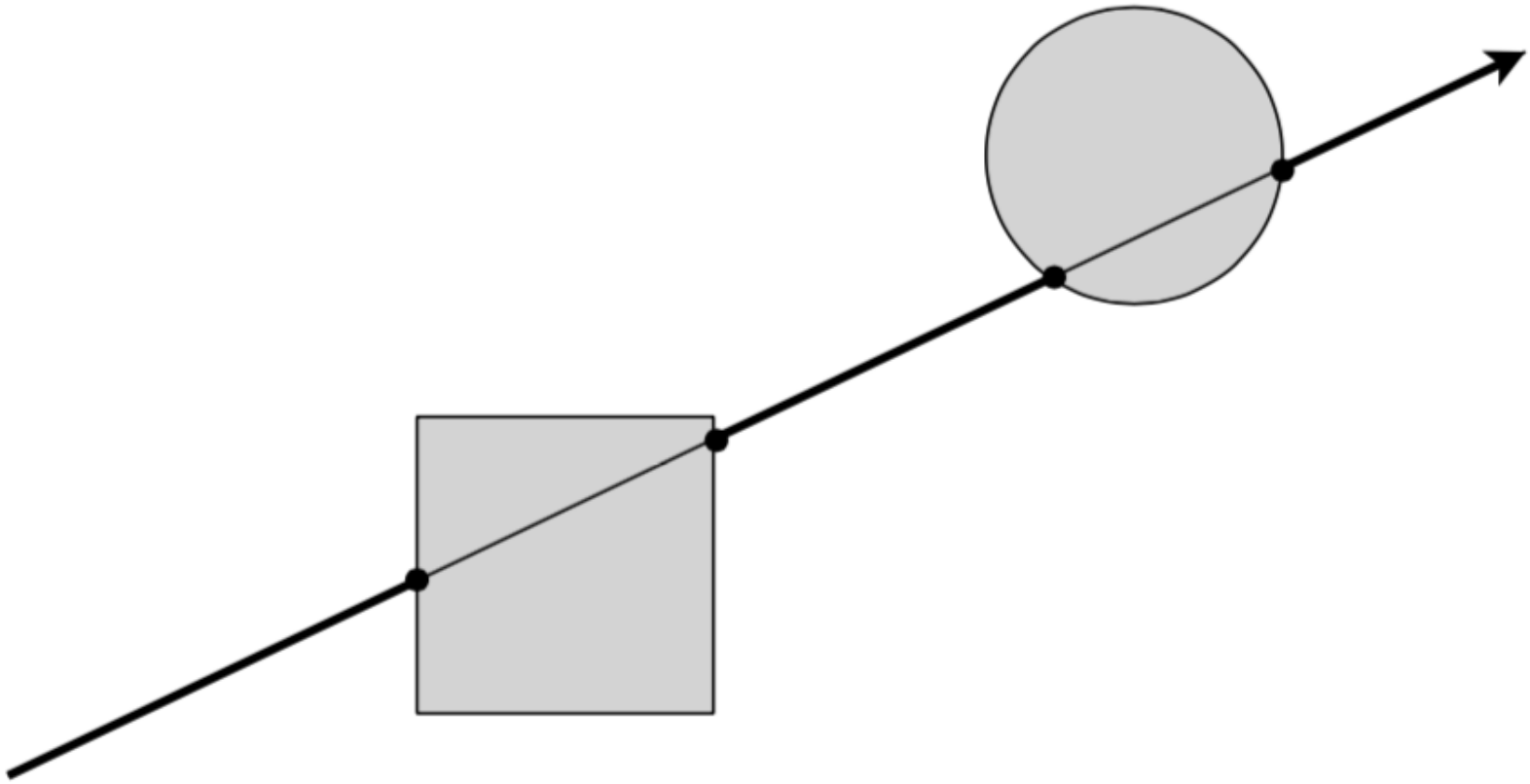
Mapeamento

- One last detail: exactly where are pixels located?



$$u = (i + 0.5)/n_x$$
$$v = (j + 0.5)/n_y$$

Intersecções

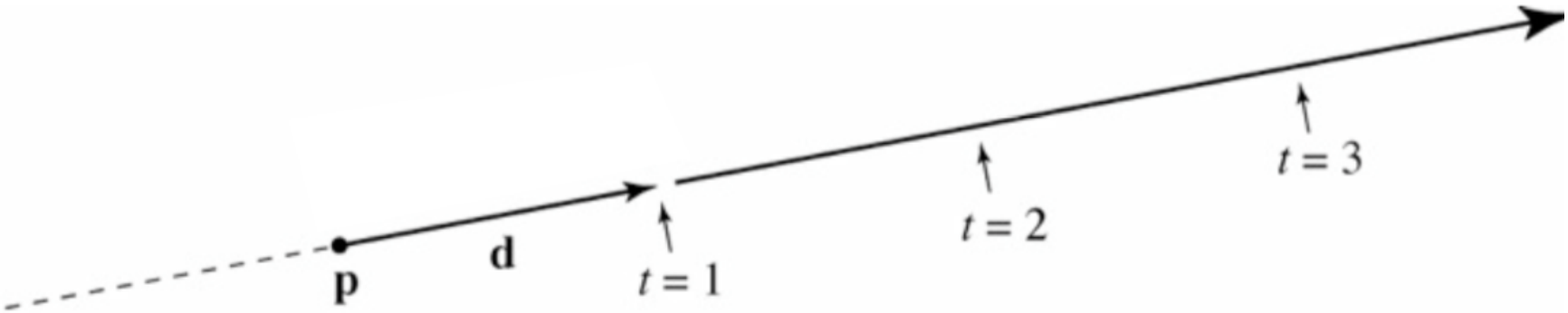


Raios

- Standard representation: point **p** and direction **d**

$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$

  - this is a *parametric equation* for the line
  - lets us directly generate the points on the line
  - if we restrict to $t > 0$ then we have a ray
  - note replacing **d** with $\alpha\mathbf{d}$ doesn't change ray ($\alpha > 0$)



Intersecção entre raio e esfera

- Condition 1: point is on ray

$$\mathbf{r}(t) = \mathbf{p} + t\mathbf{d}$$

- Condition 2: point is on sphere
  - assume unit sphere;

$$\|\mathbf{x}\| = 1 \Leftrightarrow \|\mathbf{x}\|^2 = 1$$

$$f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{x} - 1 = 0$$

- Substitute:

$$(\mathbf{p} + t\mathbf{d}) \cdot (\mathbf{p} + t\mathbf{d}) - 1 = 0$$

  - this is a quadratic equation in *t*

Intersecção entre raio e esfera

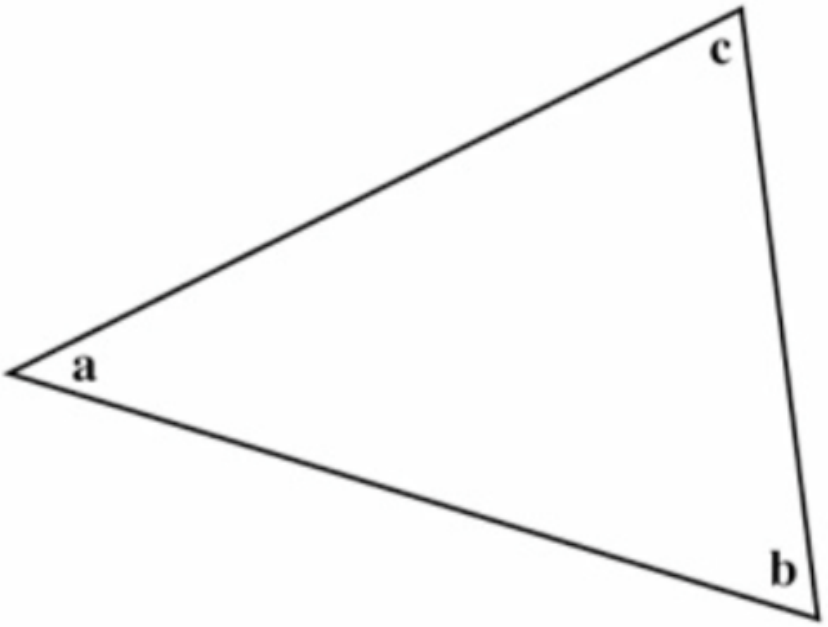
- Solution for *t* by quadratic formula:

$$t = \frac{-\mathbf{d}\cdot\mathbf{p} \pm \sqrt{(\mathbf{d}\cdot\mathbf{p})^2 - (\mathbf{d}\cdot\mathbf{d})(\mathbf{p}\cdot\mathbf{p}-1)}}{\mathbf{d}\cdot\mathbf{d}}$$

$$t = -\mathbf{d}\cdot\mathbf{p} \pm \sqrt{(\mathbf{d}\cdot\mathbf{p})^2 - \mathbf{p}\cdot\mathbf{p} + 1}$$

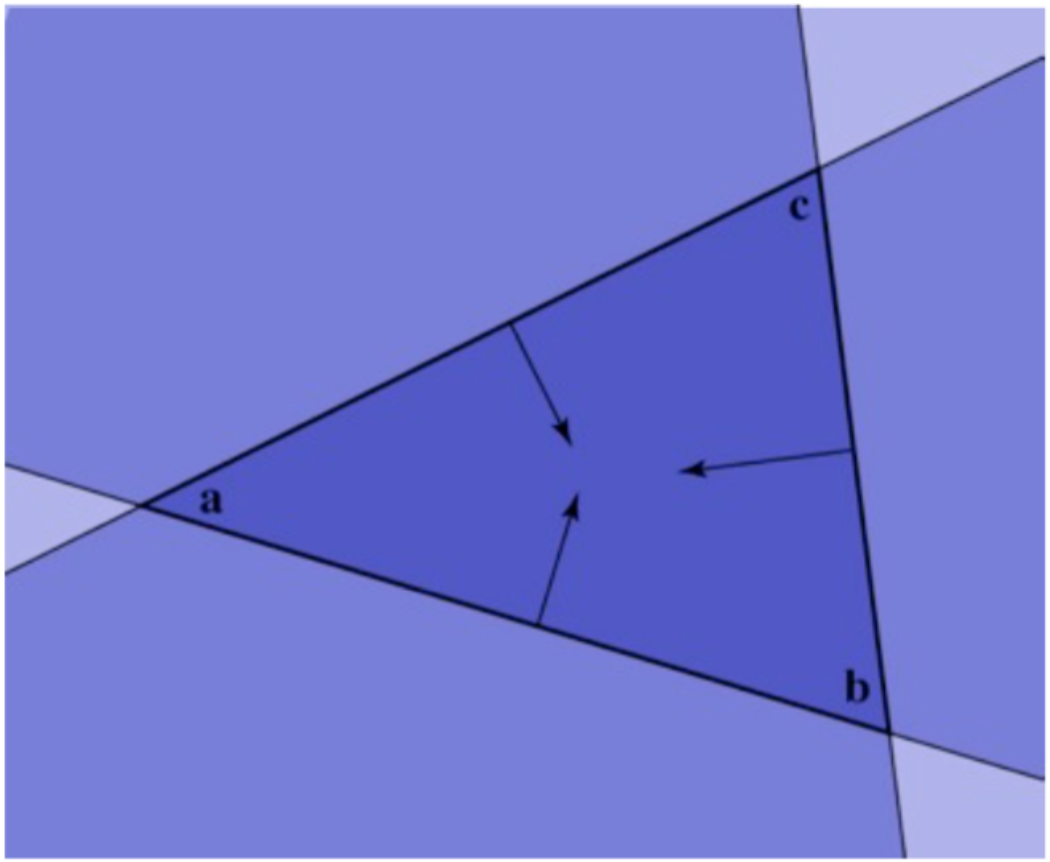
- Check discriminant for determining the number of solutions

Intersecção entre raio e triângulo



Intersecção entre raio e triângulo

- In plane, triangle is the intersection of 3 half spaces



Ponto pertencente à triângulo

- Need to check whether hit point is inside 3 edges – easiest to do in 2D coordinates on the plane
- Will also need to know where we are in the triangle – for textures, shading, etc.
- Efficient solution: transform to coordinates aligned to the triangle

Coordenadas baricêntricas

A coordinate system for triangles

- algebraic viewpoint:

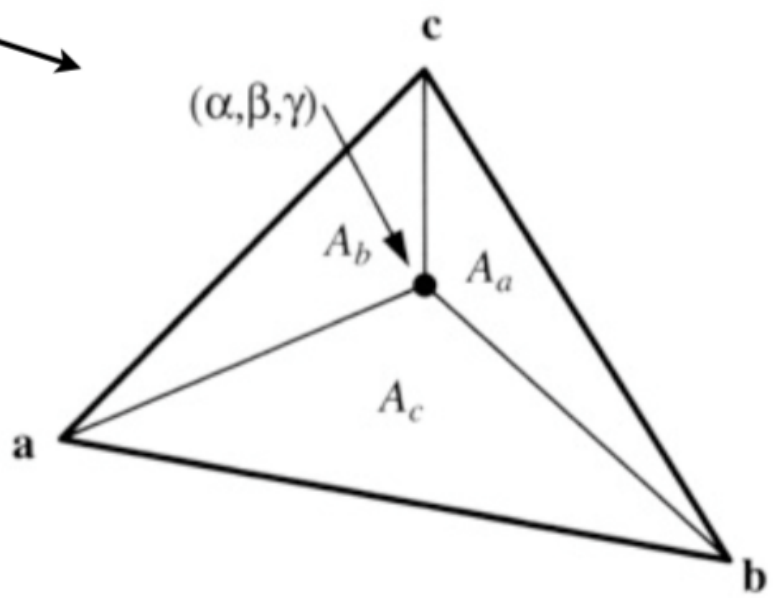
$$\mathbf{p} = \alpha\mathbf{a} + \beta\mathbf{b} + \gamma\mathbf{c}$$

$$\alpha + \beta + \gamma = 1$$

- geometric viewpoint (areas):

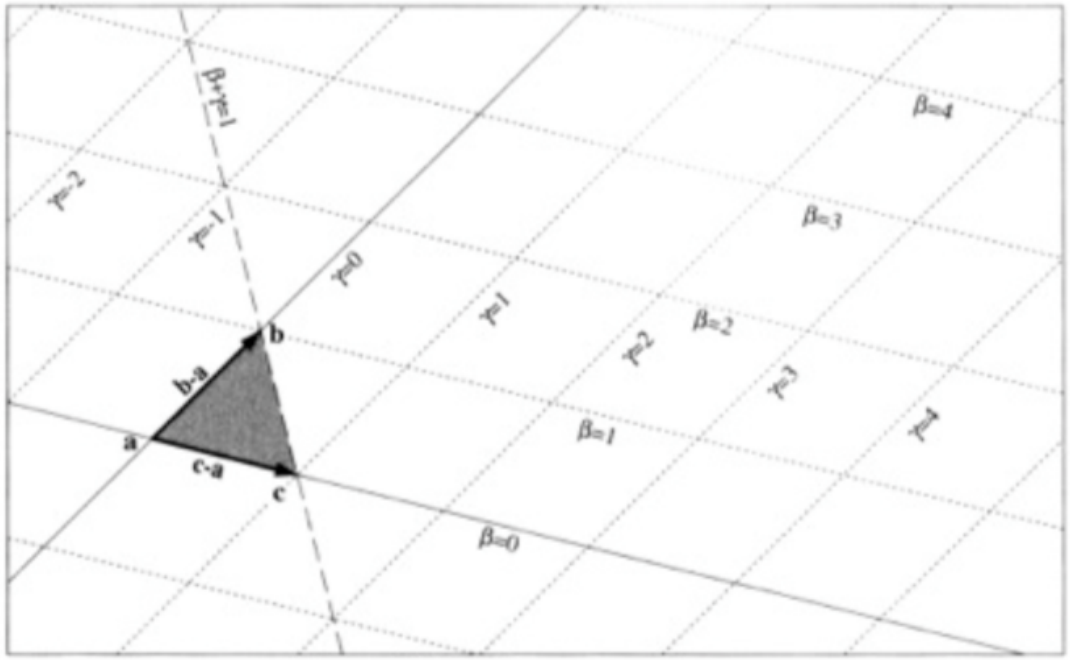
Triangle interior test:

$$\alpha > 0; \quad \beta > 0; \quad \gamma > 0$$



Coordenadas baricêntricas

- Linear viewpoint: basis for the plane



– in this view, the triangle interior test is just

$$\beta > 0; \quad \gamma > 0; \quad \beta + \gamma < 1$$

Intersecção entre raio e triângulo

- Every point on the plane can be written in the form:

$$\mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

for some numbers β and γ.

- If the point is also on the ray then it is

$$\mathbf{p} + t\mathbf{d}$$

for some number t.

- Set them equal: 3 linear equations in 3 variables

$$\mathbf{p} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

…solve them to get t, β, and γ all at once!

Intersecção entre raio e triângulo

$$\mathbf{p} + t\mathbf{d} = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

$$\beta(\mathbf{a} - \mathbf{b}) + \gamma(\mathbf{a} - \mathbf{c}) + t\mathbf{d} = \mathbf{a} - \mathbf{p}$$

$$\begin{bmatrix} \mathbf{a} - \mathbf{b} & \mathbf{a} - \mathbf{c} & \mathbf{d} \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} \mathbf{a} - \mathbf{p} \end{bmatrix}$$

$$\begin{bmatrix} x_a - x_b & x_a - x_c & x_d \\ y_a - y_b & y_a - y_c & y_d \\ z_a - z_b & z_a - z_c & z_d \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} x_a - x_p \\ y_a - y_p \\ z_a - z_p \end{bmatrix}$$

Intersecção de raios em software

- All surfaces need to be able to intersect rays with themselves.

```
class Surface {
    ...
    abstract boolean intersect(IntersectionRecord result, Ray r);
}
```

ray to be intersected

was there an intersection?

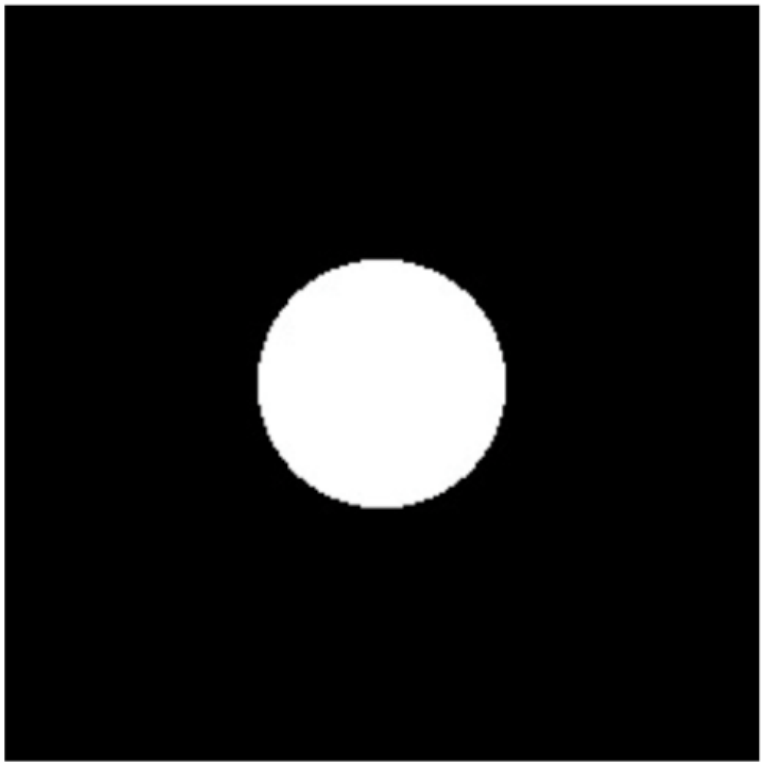
information about first intersection

```
class IntersectionRecord {
    float t;
    Vector3 hitLocation;
    Vector3 normal;
    ...
}
```

Imagem resultante até o momento

- With eye ray generation and sphere intersection

```
Surface s = new Sphere((0.0, 0.0, 0.0), 1.0);
for 0 <= iy < ny
    for 0 <= ix < nx {
        ray = camera.getRay(ix, iy);
        hitSurface, t = s.intersect(ray, 0, +inf)
        if hitSurface is not null
            image.set(ix, iy, white);
    }
```



Intersecção de raios em software

- Scenes usually have many objects
- Need to find the first intersection along the ray
  - that is, the one with the smallest positive t value
- Loop over objects
  - ignore those that don't intersect
  - keep track of the closest seen so far
  - Convenient to give rays an ending t value for this purpose

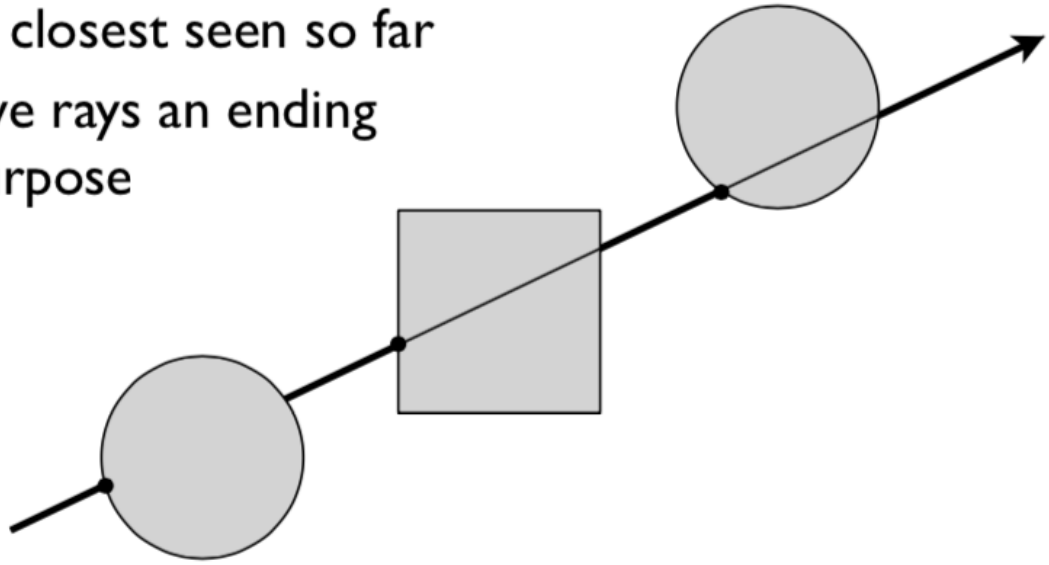


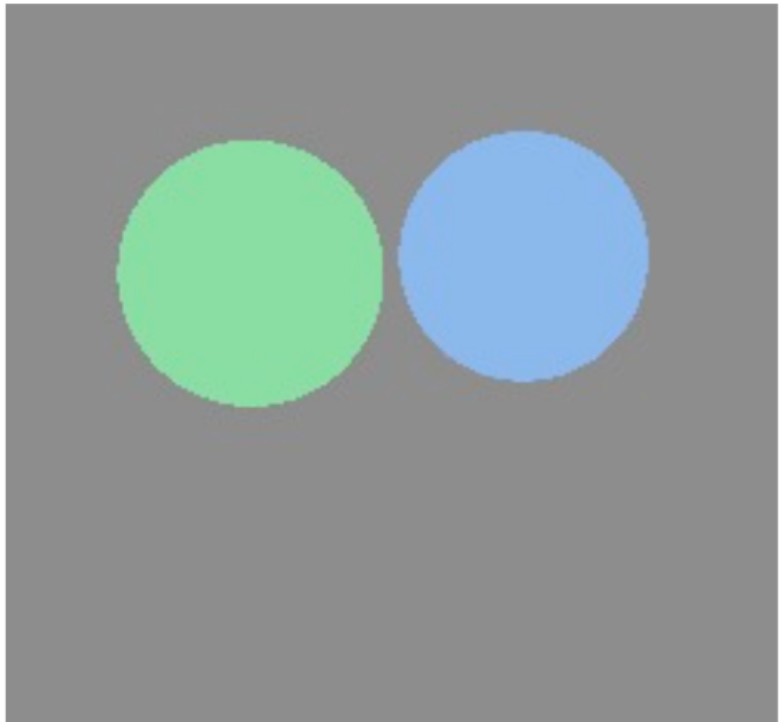
Imagem obtida até o momento

- With eye ray generation and scene intersection

```
for 0 <= iy < ny
    for 0 <= ix < nx {
        ray = camera.getRay(ix, iy);
        c = scene.trace(ray, 0, +inf);
        image.set(ix, iy, c);
    }

...

Scene.trace(ray, tMin, tMax) {
    surface, t = surfs.intersect(ray, tMin, tMax);
    if (surface != null) return surface.color();
    else return black;
}
```

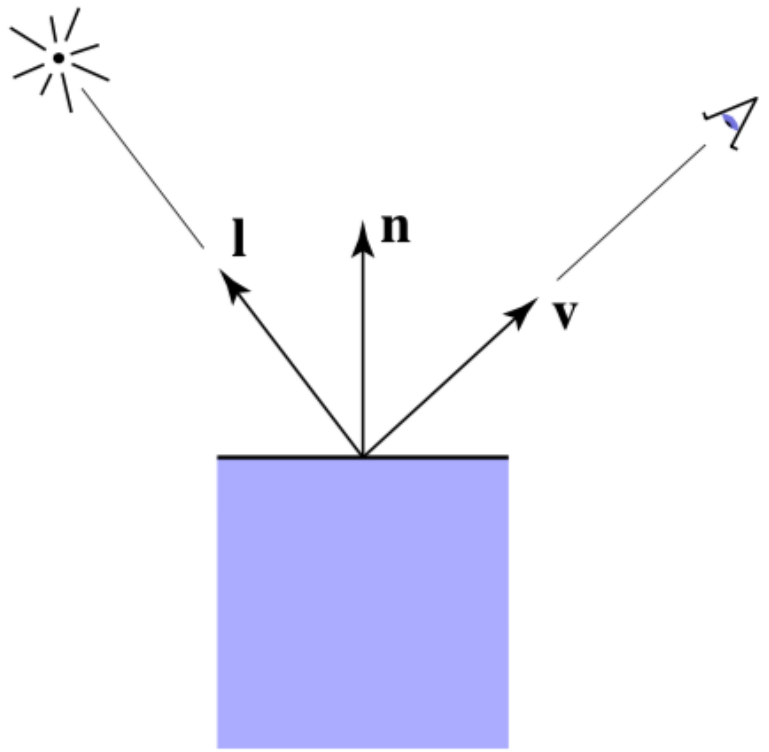


Tonalização

Compute light reflected toward camera

Inputs:

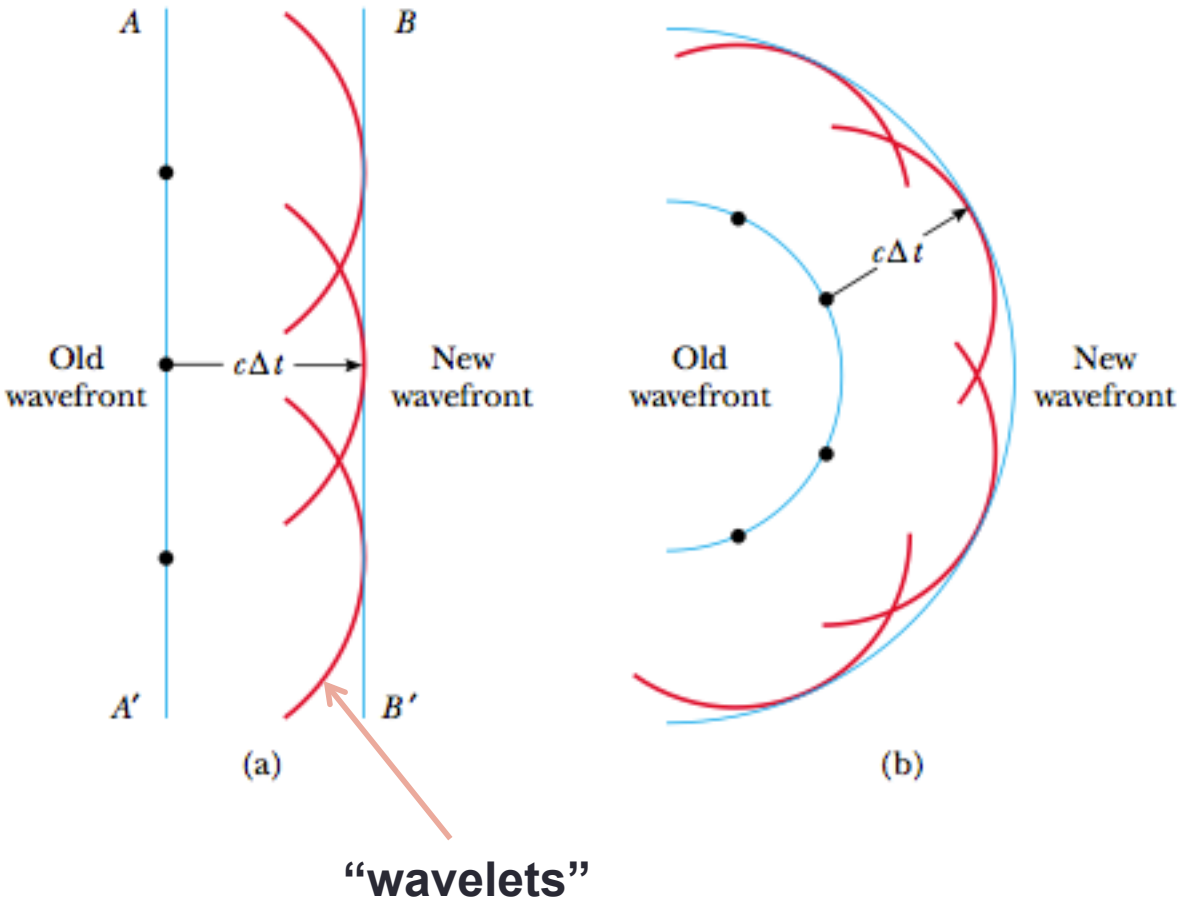
- eye direction
- light direction
  (for each of many lights)
- surface normal
- surface parameters
  (color, shininess, …)



Aproximação por raio

- Raio é perpendicular a fronteira de propagação da onda
- Fontes são supostas pontuais
- Fendas consideradas não alteram a direção de propagação
- Modelo suficiente para estudar espelhos, lentes, prismas e instrumentos óticos associados.

Princípio de Huygens



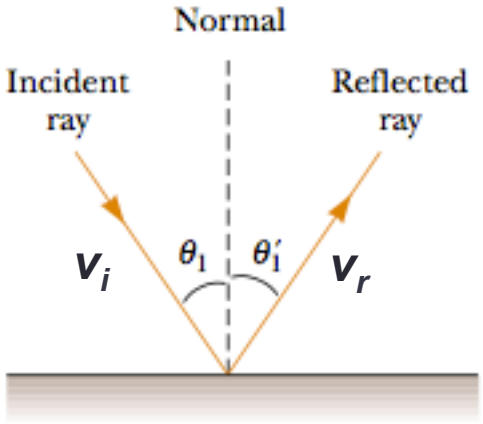
Christiaan Huygens (1629-1695), Traité de la Lumière, 1690.

Comportamentos dos raios

- Ao atingir fronteiras entre meios com características materiais distintas, alguns fenômenos mudam a direção de propagação:
  - Reflexão
  - Refração
  - Difração

Reflexão

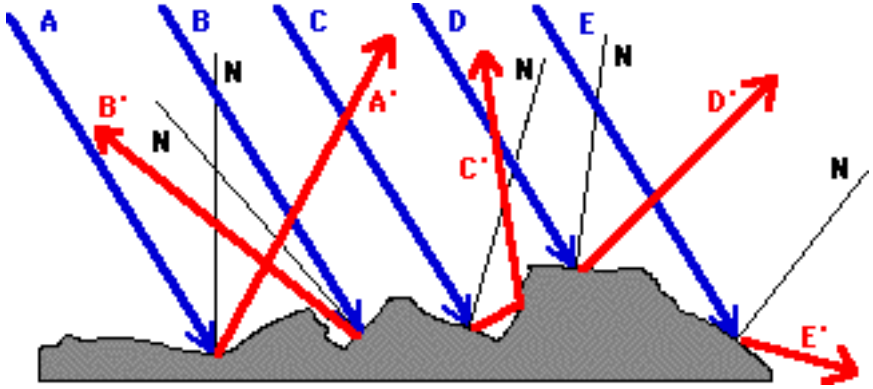
- Quando um raio de luz viajando em um meio encontra a fronteira de outro meio, parte da luz incidente é refletida

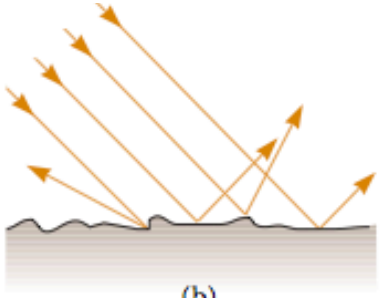


- Dados um vetor de luz incidente (v_i), como calcular o vetor de reflexão (v_r) ?

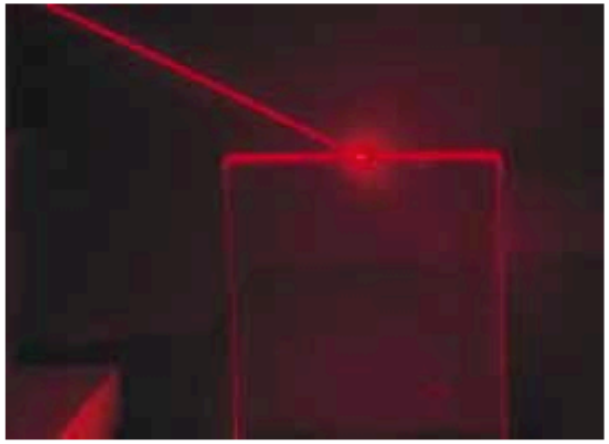
Reflexão difusa

- A maioria dos objetos, cujas superfícies são naturalmente rugosas, dão origem à reflexão difusa.





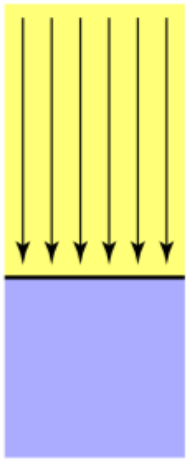
(b)



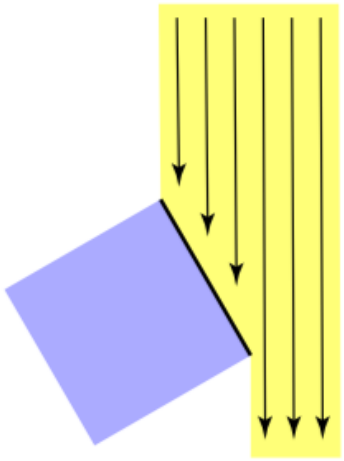
(d)

Reflexão difusa

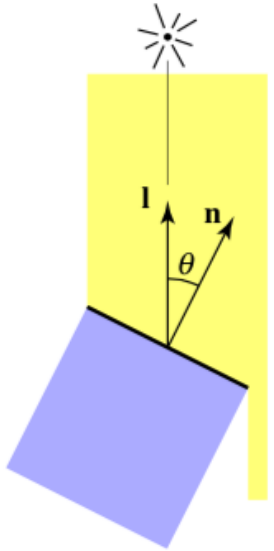
- Light is scattered uniformly in all directions
  - the surface color is the same for all viewing directions
- Lambert's cosine law



Top face of cube receives a certain amount of light

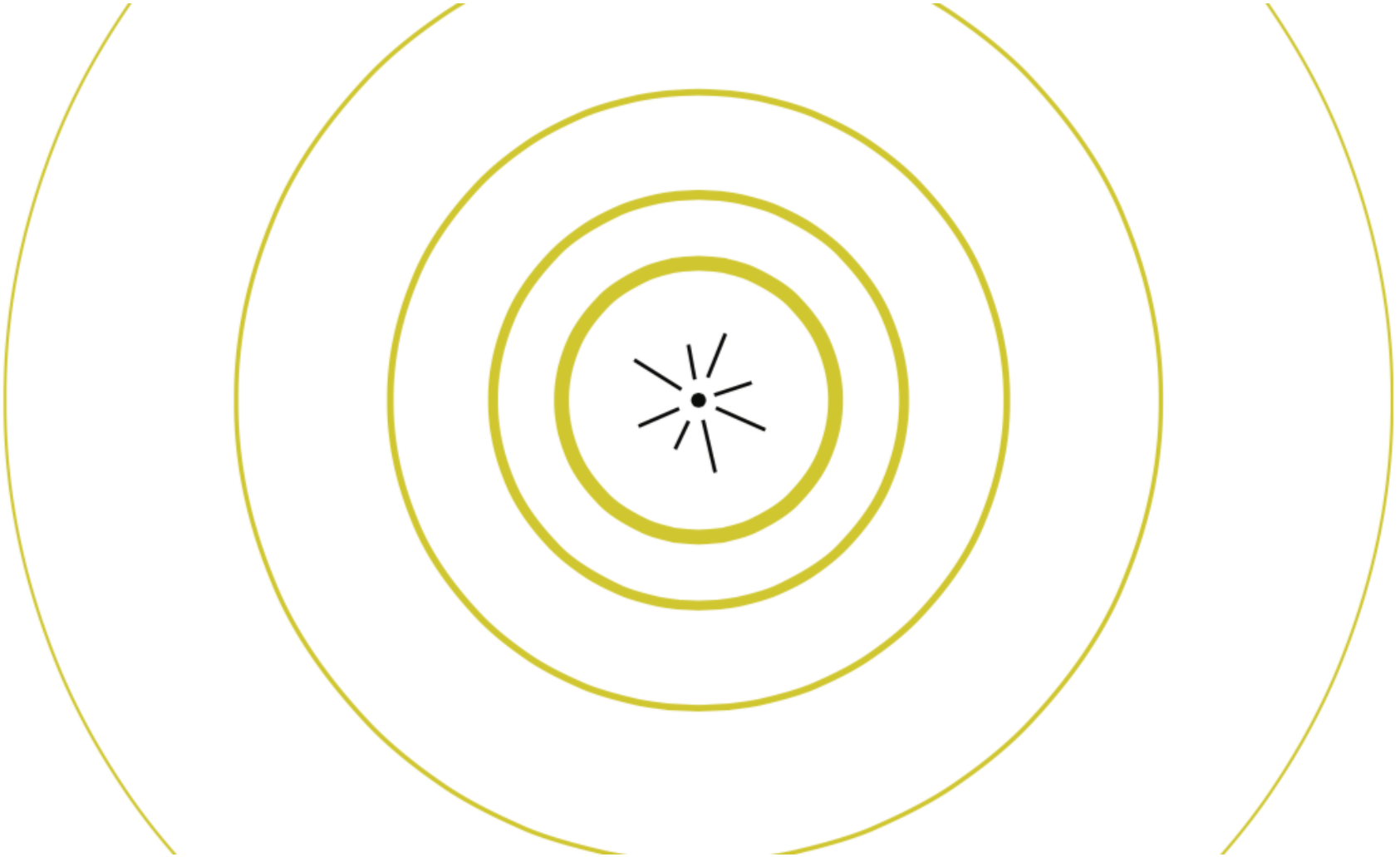


Top face of 60° rotated cube intercepts half the light

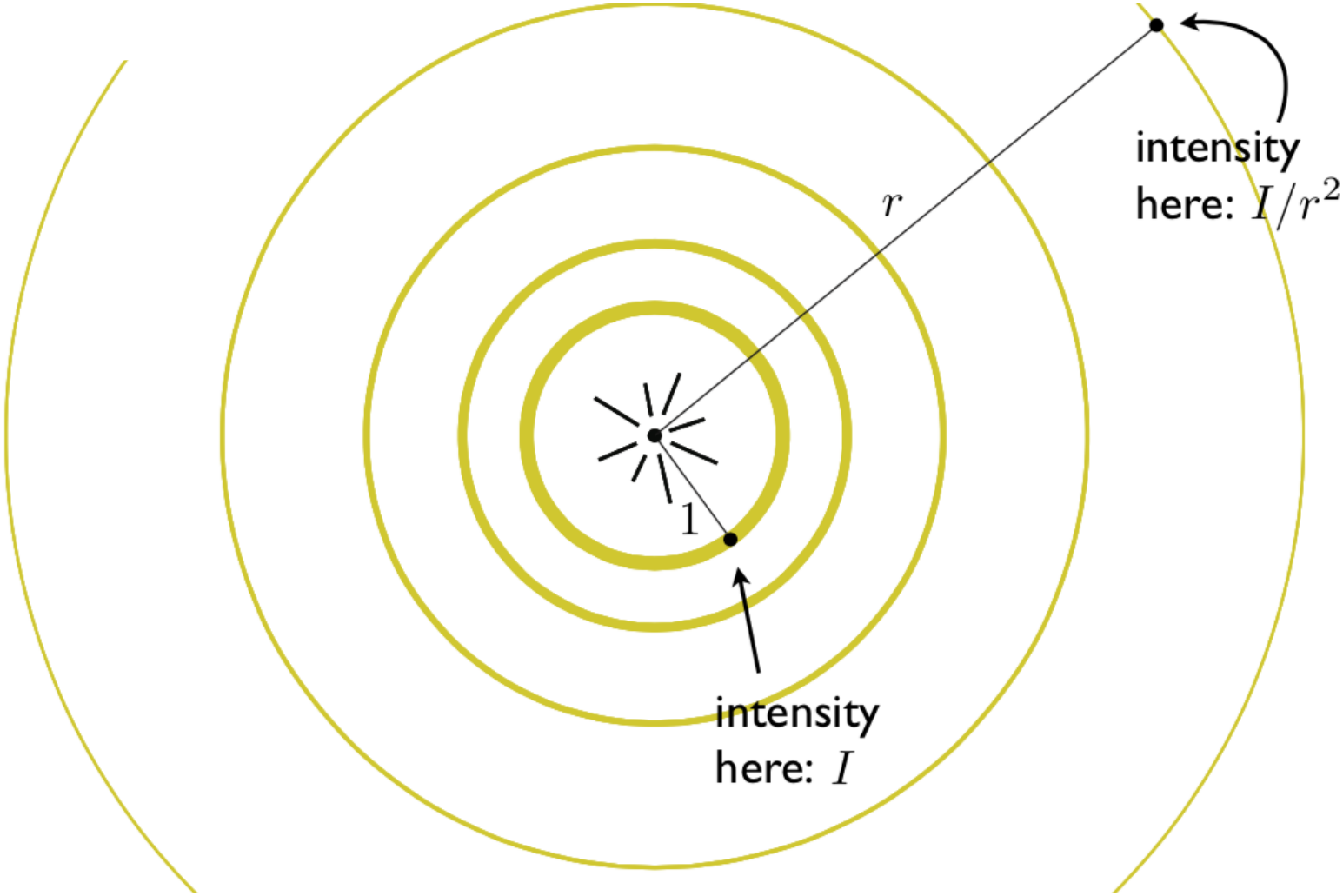


In general, light per unit area is proportional to $\cos\theta = \mathbf{l} \cdot \mathbf{n}$

Atenuação por distância

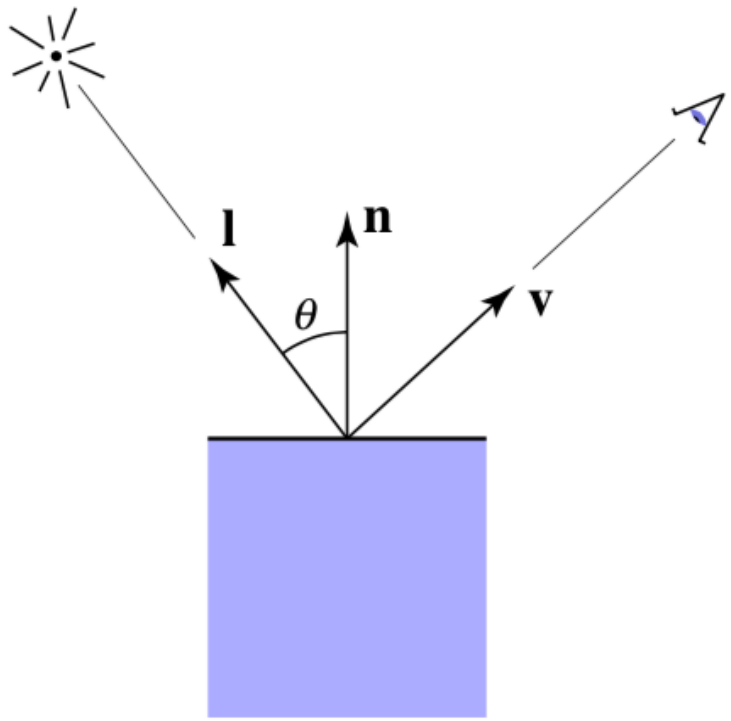


Atenuação por distância



Reflexão Lambertiana

- Shading independent of view direction



illumination from source

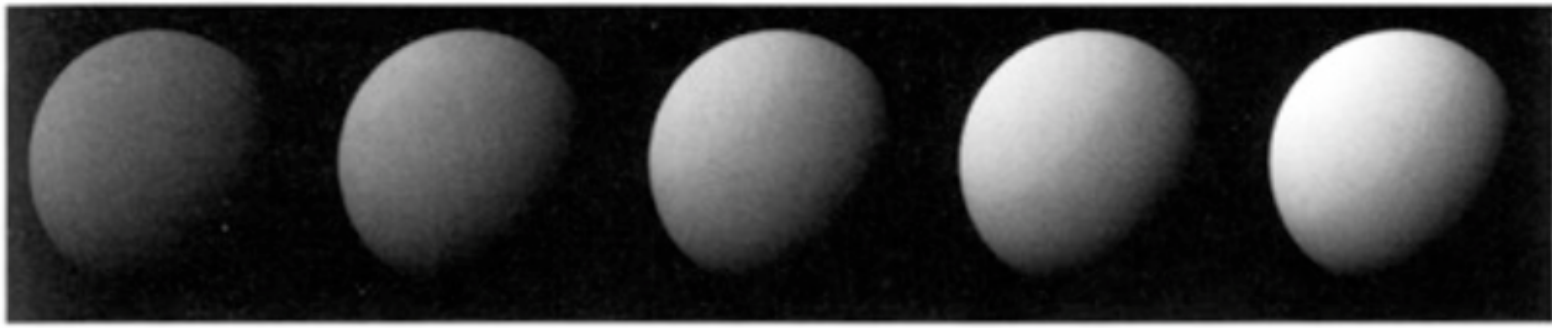
$$L_d = k_d \, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l})$$

diffuse coefficient

diffusely reflected light

Tonalização difusa

- Produces matte appearance



k_d ⟶

Resultado esperado

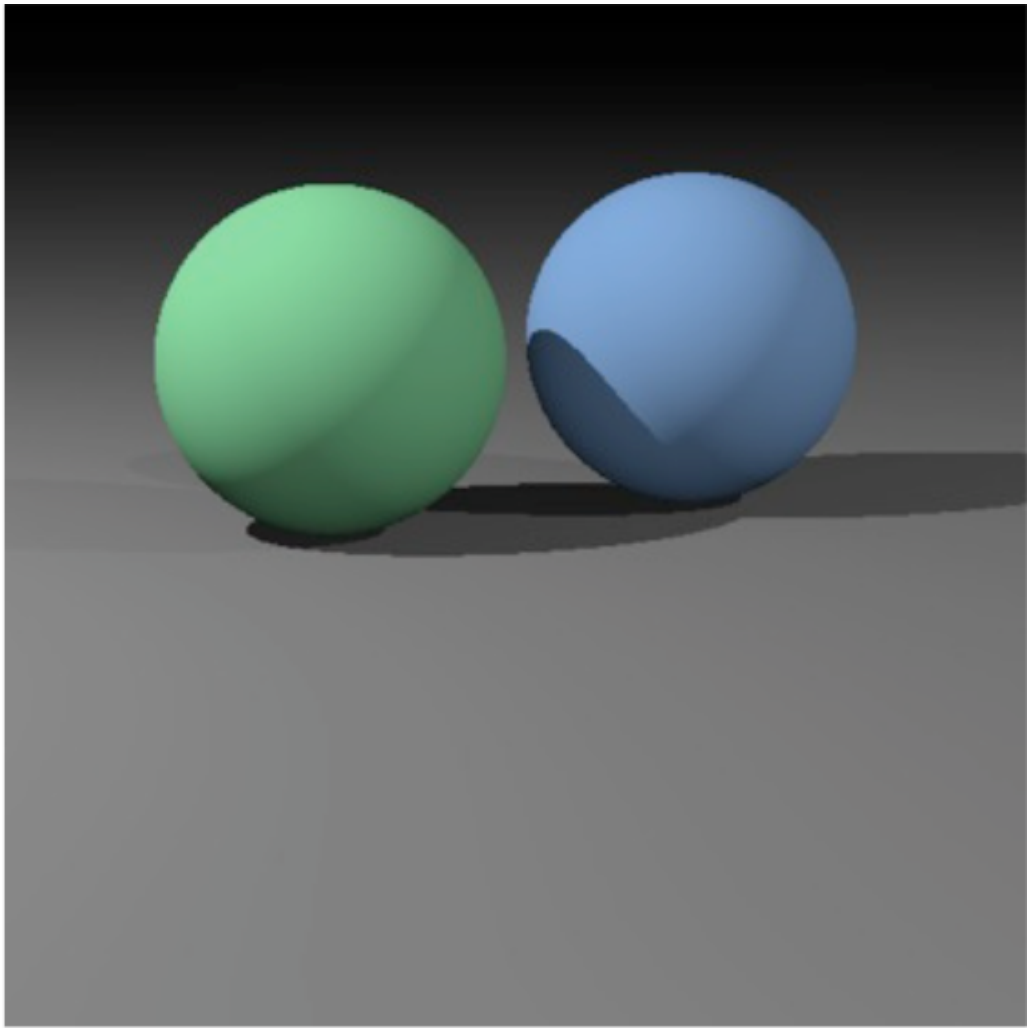
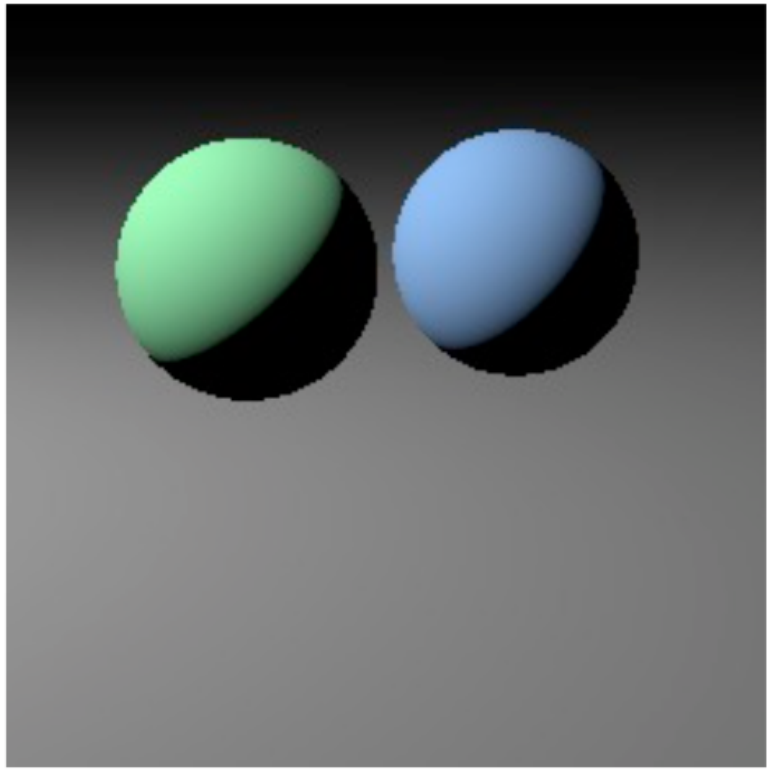


Imagem obtida até o momento

```
Scene.trace(Ray ray, tMin, tMax) {
    surface, t = hit(ray, tMin, tMax);
    if surface is not null {
        point = ray.evaluate(t);
        normal = surface.getNormal(point);
        return surface.shade(ray, point,
            normal, light);
    }
    else return backgroundColor;
}

...

Surface.shade(ray, point, normal, light) {
    v = −normalize(ray.direction);
    l = normalize(light.pos − point);
    // compute shading
}
```

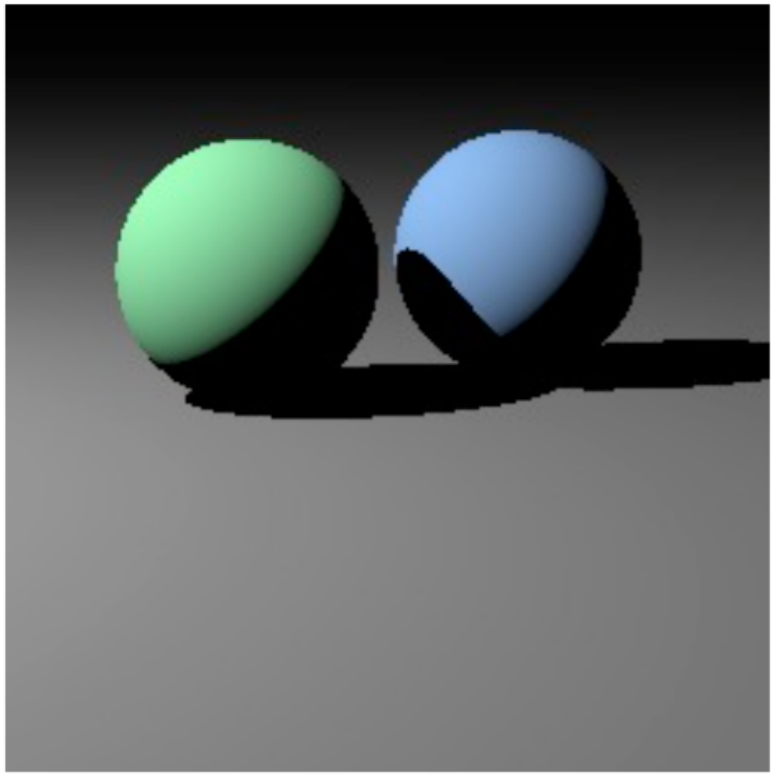


Sombras

- Surface is only illuminated if nothing blocks its view of the light
- With ray tracing it's easy to check – just intersect a ray with the scene!
  - Cast shadow rays

Imagem obtida até o momento

```
Surface.shade(ray, point, normal, light) {
    shadRay = (point, light.pos − point);
    if (shadRay not blocked) {
        v = −normalize(ray.direction);
        l = normalize(light.pos − point);
        // compute shading
    }
    return black;
}
```

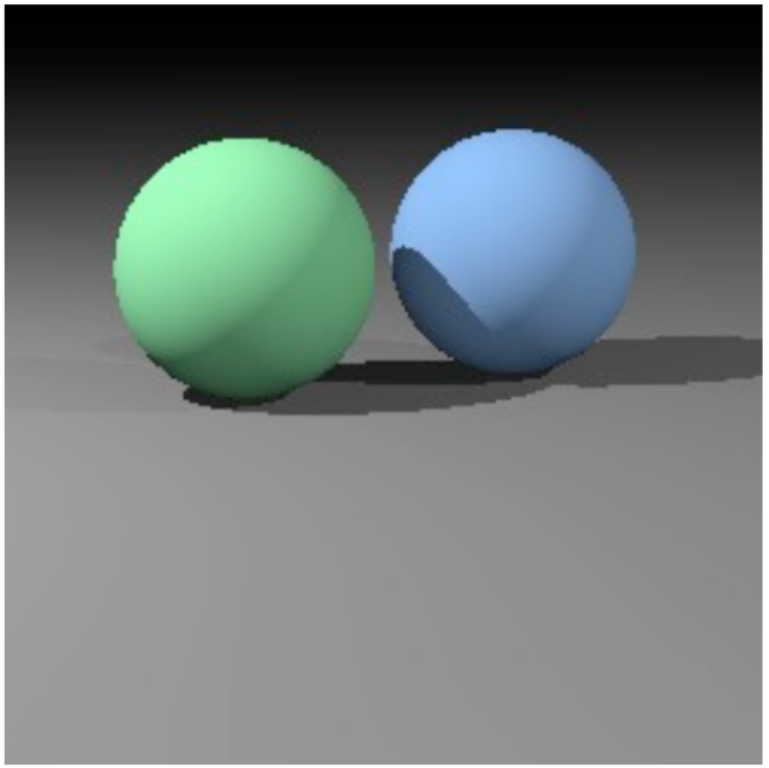


Múltiplas luzes

- Important to fill in black shadows
- Just loop over lights, add contributions
- Ambient shading
  - Black shadows are not really right
  - One solution: dim light at camera
  - Alternative: add a constant "ambient" color to the shading...

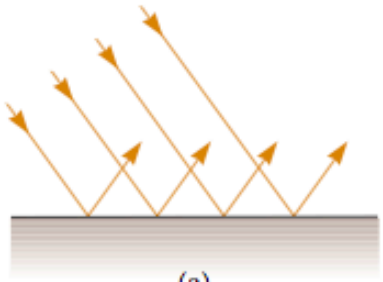
Imagem obtida até o momento

```
shade(ray, point, normal, lights) {
    result = ambient;
    for light in lights {
        if (shadow ray not blocked) {
            result += shading contribution;
        }
    }
    return result;
}
```

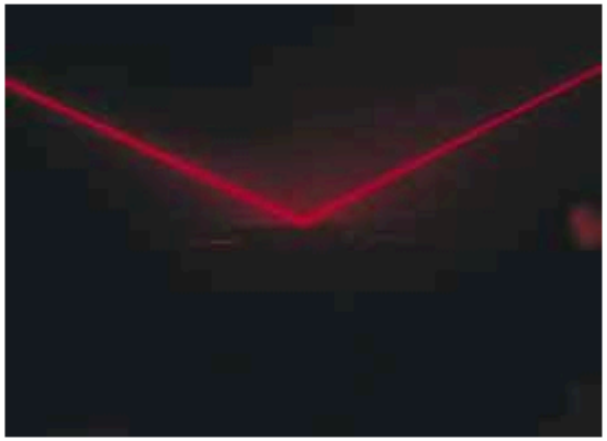


Reflexão especular

- Em materiais microscopicamente regulares (i.e. espelhos, água), a reflexão é especular.



(a)

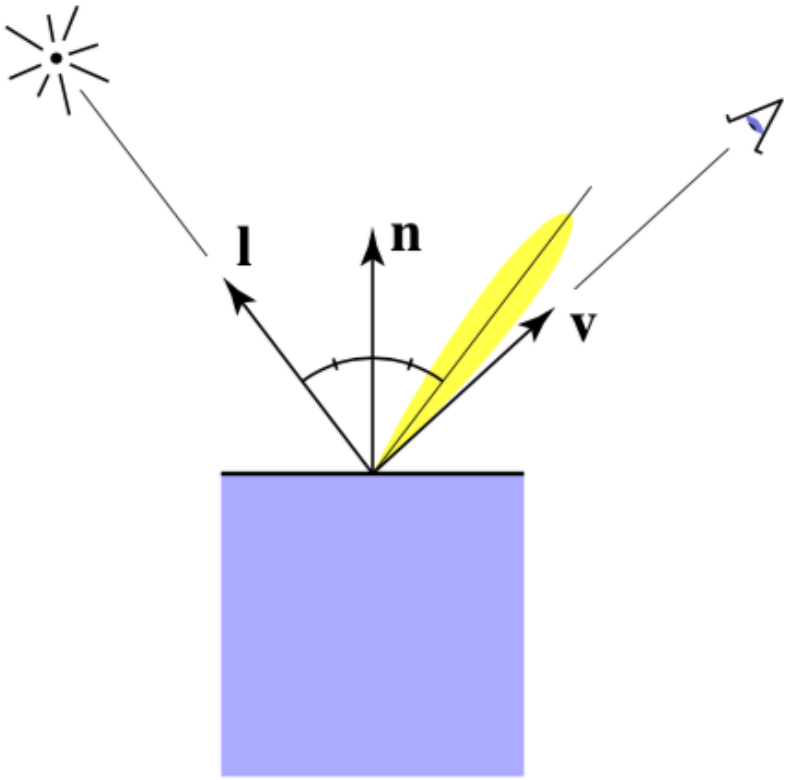


Courtesy of Henry Leap and Jim Lehman

(c)

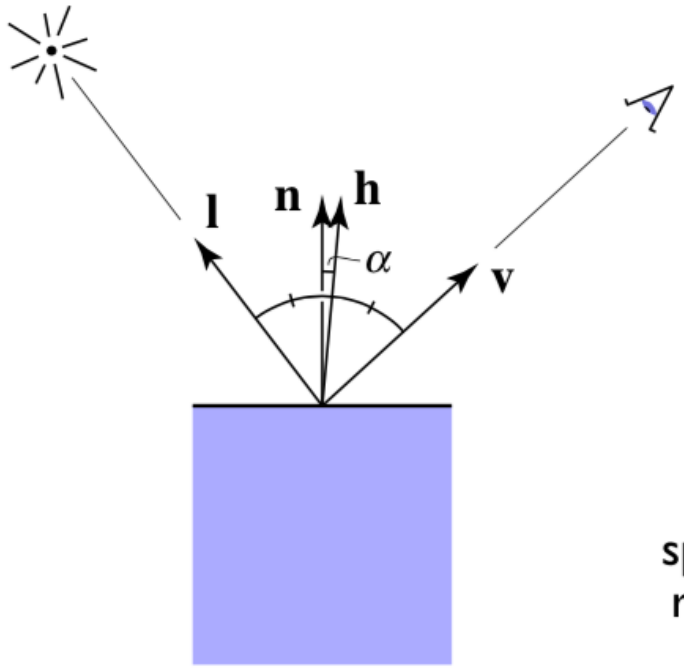
Modelo Blinn-Phong

- Intensity depends on view direction
  - bright near mirror configuration



Modelo Blinn-Phong

- Close to mirror ⇔ half vector near normal
  - Measure "near" by dot product of unit vectors



$$\mathbf{h} = \text{bisector}(\mathbf{v}, \mathbf{l}) = \frac{\mathbf{v} + \mathbf{l}}{\|\mathbf{v} + \mathbf{l}\|}$$

$$L_s = k_s \, (I/r^2) \max(0, \cos\alpha)^p = k_s \, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

specularly reflected light

specular coefficient

Modelo Phong/Blinn-Phong

- Increasing *n* narrows the lobe

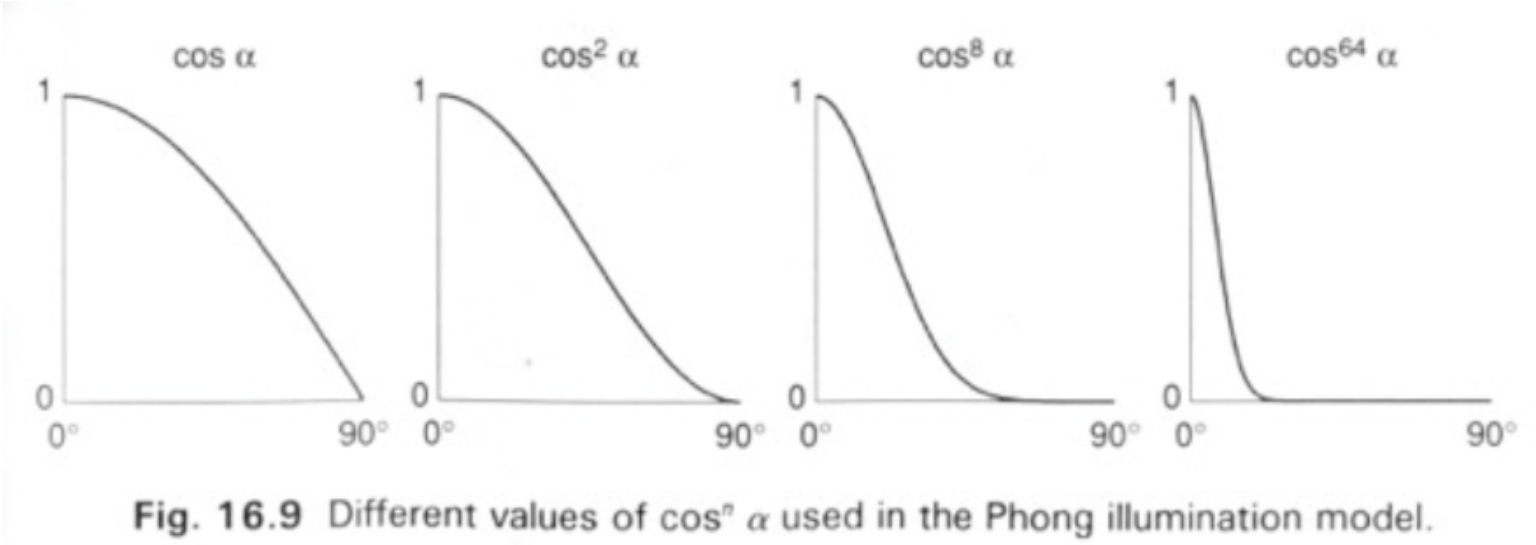
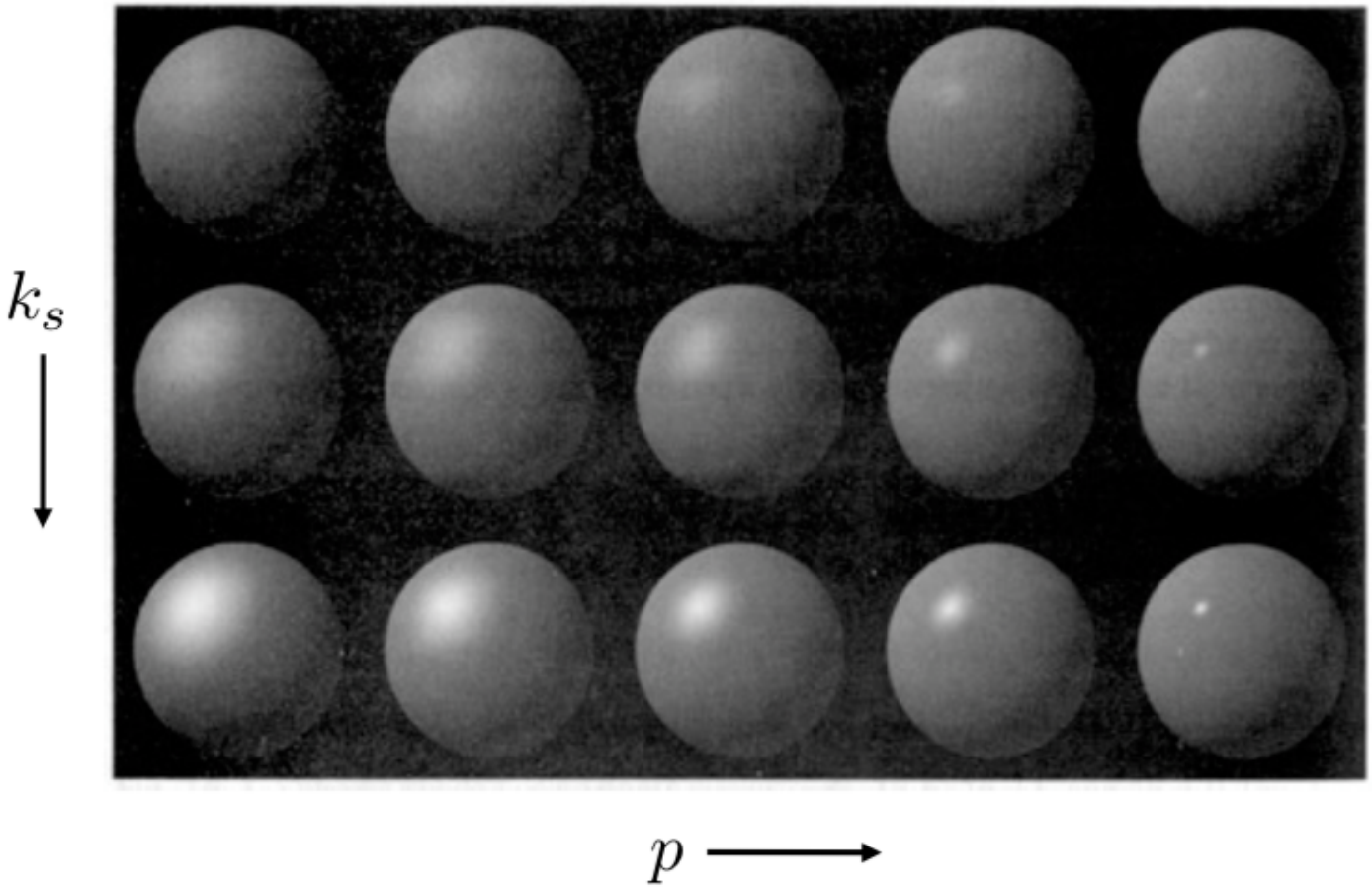
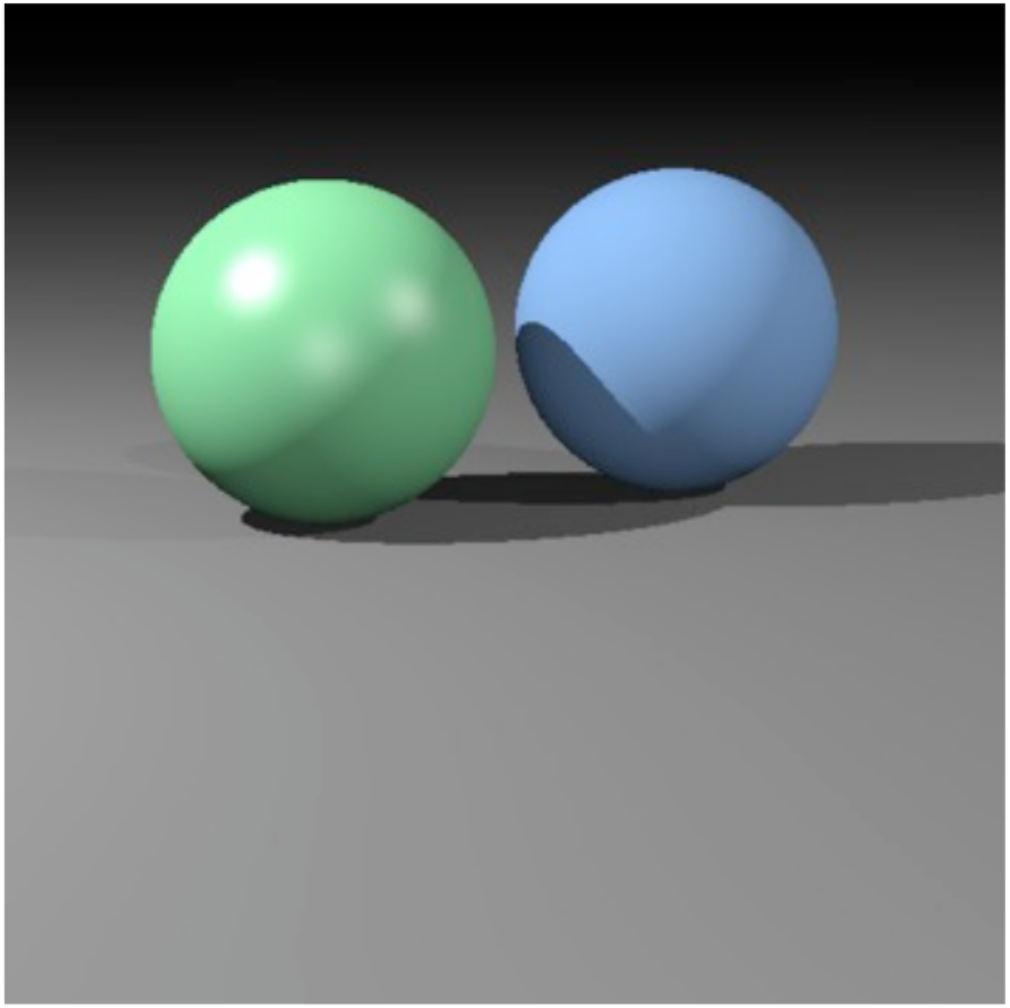


Fig. 16.9 Different values of $\cos^n \alpha$ used in the Phong illumination model.

Tonalização especular

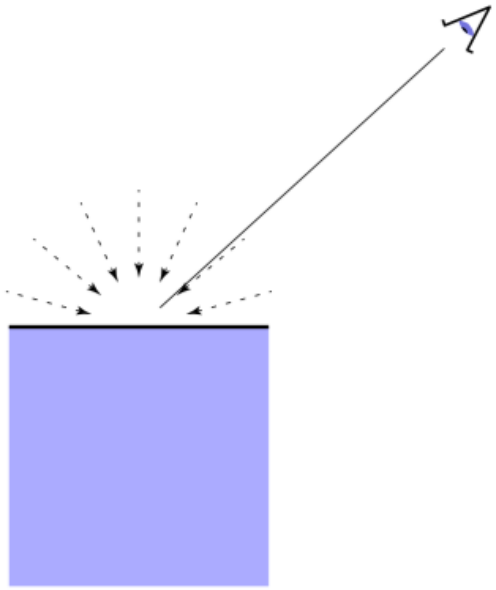


Tonalização difusa + especular



Iluminação ambiente

- Shading that does not depend on anything
  - add constant color to account for disregarded illumination and fill in black shadows



$$L_a = k_a I_a$$

ambient coefficient

reflected ambient light

Somando as contribuições

- Usually include ambient, diffuse, Phong in one model

$$L = L_a + L_d + L_s$$
$$= k_a\, I_a + k_d\, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{l}) + k_s\, (I/r^2) \max(0, \mathbf{n} \cdot \mathbf{h})^p$$

- The final result is the sum over many lights

$$L = L_a + \sum_{i=1}^{N} [(L_d)_i + (L_s)_i]$$

$$L = k_a\, I_a + \sum_{i=1}^{N} \left[k_d\, (I_i/r_i^2) \max(0, \mathbf{n} \cdot \mathbf{l}_i) + \right.$$
$$\left. k_s\, (I_i/r_i^2) \max(0, \mathbf{n} \cdot \mathbf{h}_i)^p\right]$$

Tarefa

- Leitura livro-texto
  - Shirley and Marschner. Fundamentals of Computer Graphics, CRC Press, 3rd Ed. 2010
  - Capítulos 2-4