Traçado de Raios

INF2604 – Fundamentos de Computação Gráfica

Waldemar Celes
celes@inf.puc-rio.br

Departamento de Informática, PUC-Rio

Algoritmo de traçado de raios

Iluminação direta (local)

- Procedimento
  - Geração dos raios
  - Cálculo de interseção com os objetos da cena
  - Cálculo de iluminação (*shading*)

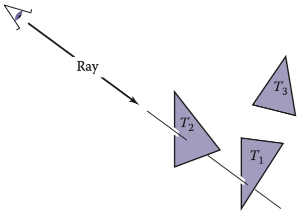


Imagem extraída de Fundamentals of Computer Graphics, S. Marchner and P. Shirley, 2016

Algoritmo de traçado de raios

Iluminação direta (local)

- Procedimento
  - Geração dos raios
  - Cálculo de interseção com os objetos da cena
  - Cálculo de iluminação (*shading*)

Pseudo-código:

for each pixel **do**
  $\vec{\mathbf{r}}$ = generate viewing ray
  $\mathbf{p}$ = find first intersection point
  $\mathbf{c}$ = compute pixel color

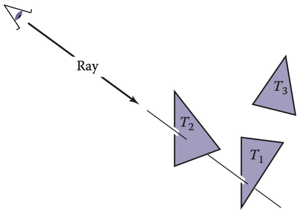
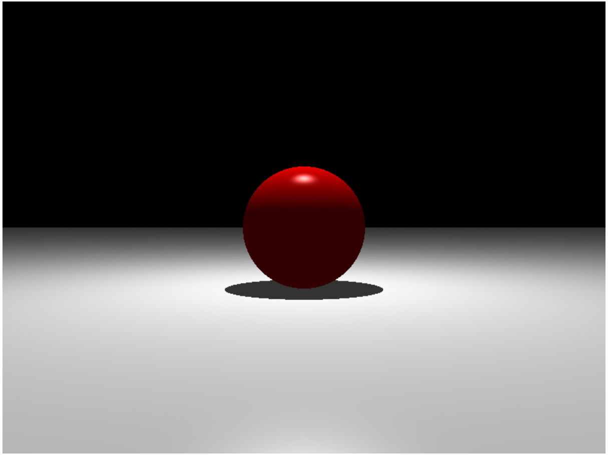


Imagem extraída de Fundamentals of Computer Graphics, S. Marchner and P. Shirley, 2016

Algoritmo de traçado de raios

Objetivo: renderizar imagem de esferas sobre plano



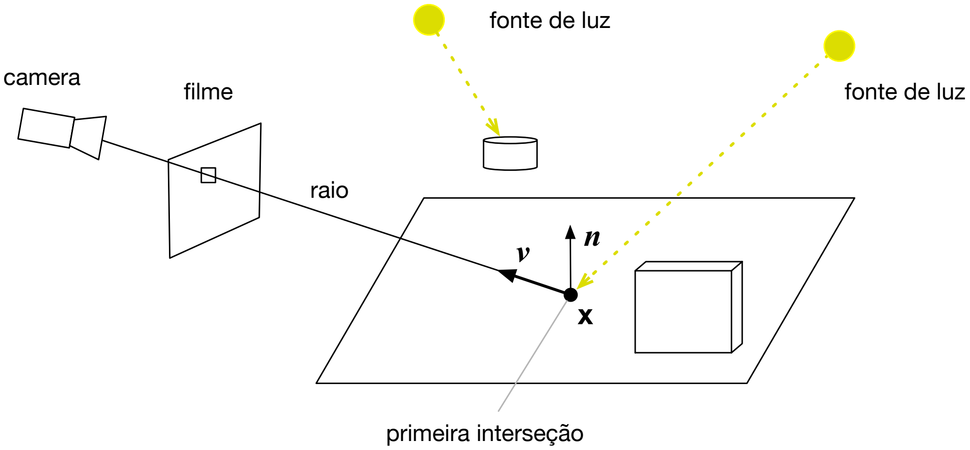
Algoritmo de traçado de raios

Tópicos

- Interseção raio-objetos (e visibilidade)
- Modelo de câmera e filme
- Espaços de coordenadas e transformações
- Geração de raios
- Fontes de luz básicas
- Modelo de iluminação local
- Instanciação de objetos
- Iluminação ambiente
- Geração de sombras
- *Antialiasing*
- Fonte de luz de área

Traçado de raios

Interação raio-cena



Interseção de raio-objetos

Raio

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

- onde:

$$\mathbf{o} = \begin{bmatrix} o_x \\ o_y \\ o_z \end{bmatrix} \qquad \hat{\mathbf{d}} = \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix}$$

- Considerando $\hat{\mathbf{d}}$ sendo um vetor unitário: t é a distância $|\mathbf{r}(t) - \mathbf{o}|$

Interseção de raio-objetos

Representação da interseção: $(\mathbf{p},\ \hat{\mathbf{n}})$

- $\mathbf{p}$: ponto de interseção
- $\hat{\mathbf{n}}$: normal à superfície no ponto de interseção

Interseção de raio-objetos

Representação da interseção: $(\mathbf{p},\ \hat{\mathbf{n}})$

- $\mathbf{p}$: ponto de interseção
- $\hat{\mathbf{n}}$: normal à superfície no ponto de interseção

Tratamento de *front/back faces*

- Estratégias
  - Normal podendo estar na mesma direção do raio
    - Determinação de *back face*: $\hat{\mathbf{d}} \cdot \hat{\mathbf{n}} < 0$
    - Iluminação de *back face* exige inverter a normal
  - Normal sempre em direção contrária ao raio
    - Determinação de *back face*: indicador explícito `isbackface`
    - Ponto de interseção tem normal invertida, se *back face*
    - Estratégia adotada no meu código (não necessariamente a melhor)

Interseção raio-objeto

Objeto plano

- Representação de um plano: $(\hat{\mathbf{n}}, \mathbf{p})$
  - Plano passando pelo ponto $\mathbf{p}$ com normal $\hat{\mathbf{n}}$
- Equação implícita

$$(\mathbf{x} - \mathbf{p}) \cdot \hat{\mathbf{n}} = 0$$

  - pois o vetor $(\mathbf{x} - \mathbf{p})$ está no plano e portanto é normal à $\hat{\mathbf{n}}$
- Interseção com raio: $\mathbf{r}(t) = \mathbf{o} + t\hat{\mathbf{d}}$

$$((\mathbf{o} + t\,\hat{\mathbf{d}}) - \mathbf{p}) \cdot \hat{\mathbf{n}} = 0$$

$$(\mathbf{o} - \mathbf{p}) \cdot \hat{\mathbf{n}} + t\,\hat{\mathbf{d}} \cdot \hat{\mathbf{n}} = 0$$

$$\therefore t = \frac{(\mathbf{o} - \mathbf{p}) \cdot \hat{\mathbf{n}}}{\hat{\mathbf{d}} \cdot \hat{\mathbf{n}}}$$

Interseção de raio-objetos

Interseção raio-plano

- Interseção pode não existir: raio paralelo ao plano
  - Denominador igual a zero: $\hat{\mathbf{d}} \cdot \hat{\mathbf{n}} = 0 \quad \therefore \quad \hat{\mathbf{d}} \perp \hat{\mathbf{n}}$
    - Na prática: $|\hat{\mathbf{d}} \cdot \hat{\mathbf{n}}| < \epsilon$
- Determinação de *back face*
  - $\hat{\mathbf{d}} \cdot \hat{\mathbf{n}} > 0$
    - Ponto de interseção tem normal invertida (a depender da estratégia)

Interseção de raio-objetos

Objeto esfera

- Representação de uma esfera: $(\mathbf{c}, r)$
  - Esfera com centro $\mathbf{c} = [x_c\ y_c\ z_c]^T$ e raio r
- Equação implícita da esfera

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 - r^2 = 0$$

  - Em forma vetorial:

$$(\mathbf{x} - \mathbf{c}) \cdot (\mathbf{x} - \mathbf{c}) - r^2 = 0$$

- Interseção raio-esfera

$$(\mathbf{o} + t\hat{\mathbf{d}} - \mathbf{c}) \cdot (\mathbf{o} + t\hat{\mathbf{d}} - \mathbf{c}) - r^2 = 0$$

  - Rearrumando:

$$(\hat{\mathbf{d}} \cdot \hat{\mathbf{d}})t^2 + 2\hat{\mathbf{d}} \cdot (\mathbf{o} - \mathbf{c})t + (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - r^2 = 0$$

Interseção de raio-objetos

Interseção raio-esfera

- Equação de segundo grau: $at^2 + bt + c = 0$
  - $a = \hat{\mathbf{d}} \cdot \hat{\mathbf{d}}, \quad b = 2\hat{\mathbf{d}} \cdot (\mathbf{o} - \mathbf{c}), \quad c = (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - r^2$

$$t = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

- Raízes da equação:
  - Nenhuma raiz real
  - Uma única raiz real
  - Duas raízes reais
    - Apenas a *menor raiz positiva* nos interessa
- Determinação de *back face*
  - Se existir uma raiz negativa: $x_1 < 0$

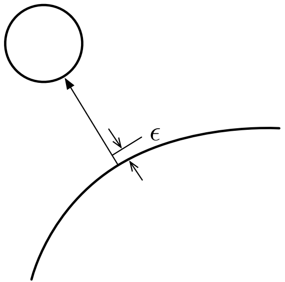
Interseção de raio-objetos

Evitando auto-interseção

- Alternativas
  - Despreza distância menores que uma tolerância

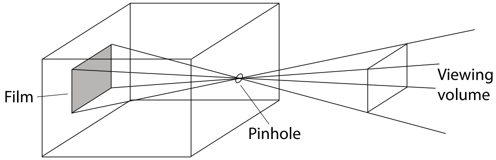
$$t > \epsilon$$

  - Despreza instância de origem
    - Instância não considerada como parte da cena



Modelo de câmera

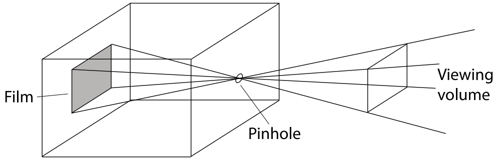
- Câmera *pinhole*



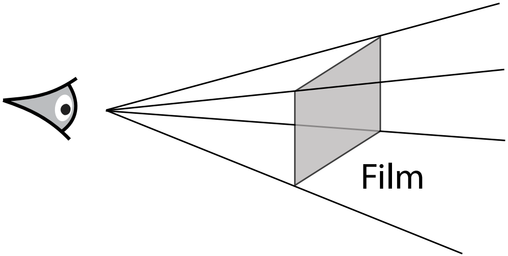
Imagens extraídas de Physically Based Rendering, Pharr et al. (www.pbr-book.org)

Modelo de câmera

- Câmera *pinhole*



- Filme na frente

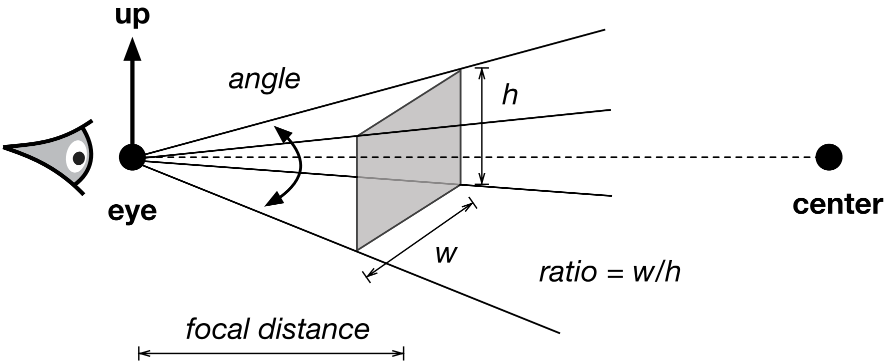


Imagens extraídas de Physically Based Rendering, Pharr et al. (www.pbr-book.org)

Modelo de câmera

Representação da câmera

- Parâmetros extrínsecos: $(\mathbf{e}, \mathbf{c}, \vec{\mathbf{v}}_{up})$
- Parâmetros intrínsecos: $(f, \theta, w/h)$



Modelo de câmera

Dados $\mathbf{e}, \mathbf{c}, \vec{\mathbf{v}}_{up}$, como construir uma base ortonormal?

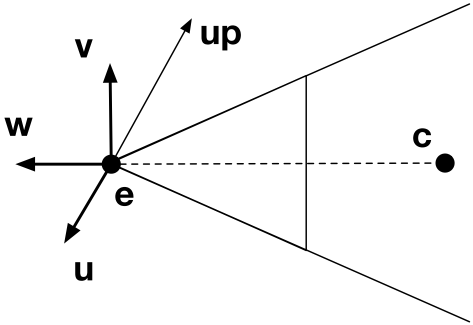
- $\vec{\mathbf{v}}_{up}$ não necessariamente perpendicular a $\mathbf{e} - \mathbf{c}$

Modelo de câmera

Dados $\mathbf{e}, \mathbf{c}, \vec{\mathbf{v}}_{up}$, como construir uma base ortonormal?

- $\vec{\mathbf{v}}_{up}$ não necessariamente perpendicular a $\mathbf{e} - \mathbf{c}$

$$\hat{\mathbf{w}} = normalize(\mathbf{e} - \mathbf{c})$$
$$\hat{\mathbf{u}} = normalize(\vec{\mathbf{v}}_{up} \times \hat{\mathbf{w}})$$
$$\hat{\mathbf{v}} = \hat{\mathbf{w}} \times \hat{\mathbf{u}}$$



Filme

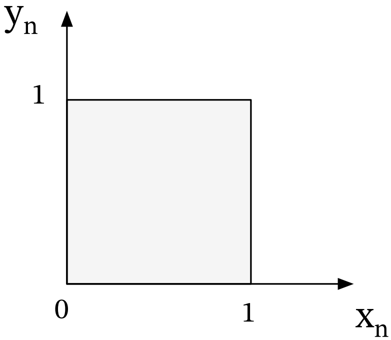
Representação de filme

- Matriz de "pixels": valores (r, g, b)
  - Resolução: $(w,\ h)$ em pixels
  - Representar canais de cor com precisão `float`
  - Converter para byte na exportação da imagem
    - Possibilita aplicação de funções de mapeamento não lineares

Filme

Geração de amostras por pixel

- Espaço local normalizado
  - Independente da resolução do filme

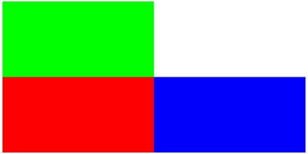


- Geração de amostras
  - Amostra no centro do pixel

$$GetSample(i, j)$$
$$\textbf{return} \left(\frac{i + 0.5}{w}, \frac{j + 0.5}{h} \right)$$

Implementação de filme

- Teste



```
#include "film.h"
const int W = 256, H = 128;
int main ()
{
  auto film = Film::Make(glm::ivec2(W,H),0);
  for (int i=0; i<W; ++i) {
    for (int j=0; j<H; ++j) {
      if (i<W/2) {
        if (j<H/2) film->SetPixelValue(i,j,glm::vec3(1.0f,0.0f,0.0f));
        else       film->SetPixelValue(i,j,glm::vec3(0.0f,1.0f,0.0f));
      }
      else {
        if (j<H/2) film->SetPixelValue(i,j,glm::vec3(0.0f,0.0f,1.0f));
        else       film->SetPixelValue(i,j,glm::vec3(1.0f,1.0f,1.0f));
      }
    }
  }
  film->SaveImage("test.jpg");
  return 0;
}
```

Implementação de filme

Método para salvar imagem: biblioteca **STB image**

```
#define STB_IMAGE_IMPLEMENTATION
#include <stb/stb_image.h>
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include <stb/stb_image_write.h>
void Film::SaveImage (const std::string& filename) const
{
  unsigned char img[3*m_resolution.x*m_resolution.y];
  // TODO: alternative mappings
  int k = 0;
  for (auto color : m_data) {
    img[k++] = (unsigned char) (color.r*255);
    img[k++] = (unsigned char) (color.g*255);
    img[k++] = (unsigned char) (color.b*255);
  }
  stbi_write_jpg(filename.c_str(),
                 m_resolution.x,m_resolution.y,3,(void*)img,100);
}
```

Geração de raio

Vale a pena gerar o raio no espaço de coordenadas global?

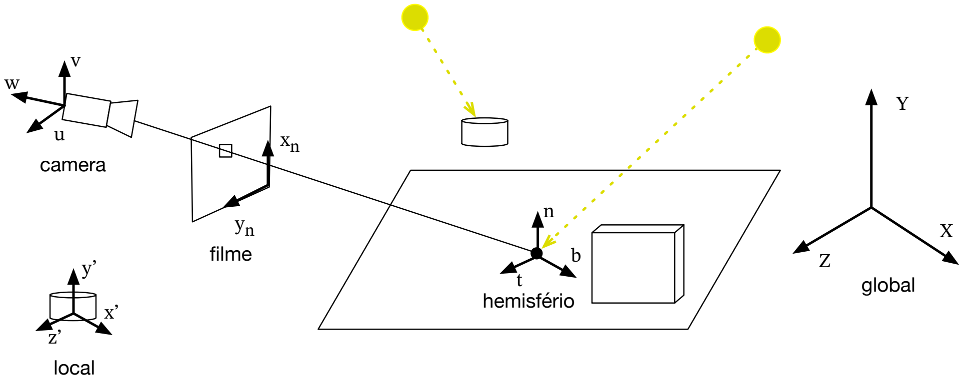
Geração de raio

Vale a pena gerar o raio no espaço de coordenadas global?

- Mais simples gerar no espaço da câmera e transformar para o global

Estruturação da cena

Espaços de coordenadas



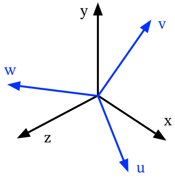
Espaços de coordenadas

Matriz mudança de base

- Matriz de rotação
- Matriz coluna dos eixos unitários

$$\mathbf{p}_{xyz} = \mathbf{B}\ \mathbf{p}_{uvw}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{u}} & \hat{\mathbf{v}} & \hat{\mathbf{w}} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$



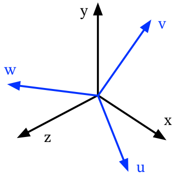
Espaços de coordenadas

Matriz mudança de base

- Matriz de rotação
- Matriz coluna dos eixos unitários

$$\mathbf{p}_{xyz} = \mathbf{B}\ \mathbf{p}_{uvw}$$

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{u}} & \hat{\mathbf{v}} & \hat{\mathbf{w}} \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$



- Transformação inversa

$$\mathbf{p}_{uvw} = \mathbf{B}^T\ \mathbf{p}_{xyz}$$

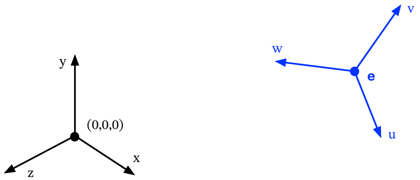
Espaços de coordenadas

Transformação entre espaços de coordenadas

- Exemplo: $xyz \longrightarrow uvw$ (espaço global para espaço da câmera)

$$\mathbf{V} = \mathbf{B}^{-\mathsf{T}}\,\mathbf{T}(-\mathbf{e})$$

onde: $\mathbf{e}$ é a posição da câmera (origem do sistema uvw expresso em xyz)



- Essa matriz 4×4 é conhecida como **matriz de visualização**
  - Trabalha-se com posições $[x\ y\ z\ w]^T$ em coordenadas homogêneas (com $w = 1$)

GLM

OpenGL Mathematics

https://github.com/g-truc/glm

- Biblioteca C++ implementada em .h apenas

```
#include <glm/glm.hpp>                        // core stuffs
#include <glm/gtc/type_ptr.hpp>               // value_ptr
#include <glm/ext/matrix_transform.hpp>       // glm::translate
                                              // glm::rotate
                                              // glm::perspective etc
#include <glm/gtx/string_cast.hpp>            // glm::to_string
```

GLM

OpenGL Mathematics

- Oferece tipos baseados em GLSL
  - `glm::vec3`, `glm::vec4`, `glm::mat4`
- Oferece funções de manipulação algébrica
  - Para vetores
    - `glm::dot(u,v)`, `glm::cross(u,v)`, `glm::lenght(v)`, `glm::distance(p,q)`
  - Para matrizes
    - `glm:transpose(M)`, `glm:inverse(M)`, `glm:translate(x,y,z)`, `glm:scale(x,y,z)`, `glm:rotate(a,x,y,z)`
  - Sobrecarga de operadores
    - `w = M * v`, `w = u * v`

Geração de raios

Estratégia

- Gera raio no espaço da câmera
- Transforma para espaço global

Geração de raios

Estratégia

- Gera raio no espaço da câmera
- Transforma para espaço global

Transformação de visualização

- Transforma do espaço global para o espaço da câmera
  - Matriz de visualização

```
glm::mat4 glm::lookAt(m_eye,m_center,m_up);
```

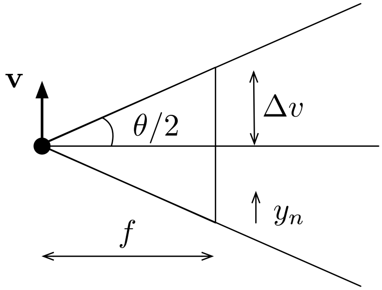
- Transformação inversa

```
glm::mat4 glm::inverse(glm::lookAt(m_eye,m_center,m_up));
```

Geração de raios

Método para geração de raio

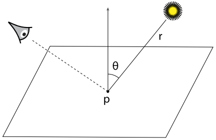
$$
\begin{aligned}
&GenerateRay(n_x, n_y)\\
&\quad \Delta v = f \tan\theta/2\\
&\quad \Delta u = \Delta v \; w/h\\
&\quad \mathbf{p} = (-\Delta u + 2\,\Delta u\, n_x, -\Delta v + 2\,\Delta v\, n_y, -f, 1)\\
&\quad \mathbf{o} = \mathbf{V}^{-1}(0,0,0,1)\\
&\quad \mathbf{t} = \mathbf{V}^{-1}\mathbf{p}\\
&\quad \textbf{return } Ray(\mathbf{o}, normalize(\mathbf{t}-\mathbf{o}))
\end{aligned}
$$

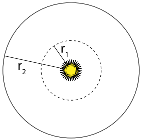


Fontes de luz básicas

Luz pontual

- Representação
  - $\mathbf{p}$: posição
  - Φ: potência
- Quantidade de luz irradiada
  - Fonte pontual emitindo energia Φ
  - Distribuição uniforme em todas as direções: $\frac{\Phi}{4\pi r^2}$





Imagens extraídas de Physically Based Rendering, Pharr et al. (www.pbr-book.org)

Fontes de luz básicas

Luz direcional (no infinito)

- Representação
  - $\mathbf{d}$: direção
  - E: irradiância
    - Decaimento desprezível

Interação luz-matéria

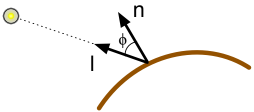
Objetivo

- Determinar a quantidade de luz irradiada do ponto em direção à câmera

Modelo de iluminação de Phong

- Componente difusa

$$\mathbf{c} = \mathbf{m}_{dif}\ \max(0, \hat{\mathbf{n}} \cdot \hat{\mathbf{l}})$$



Interação luz-matéria

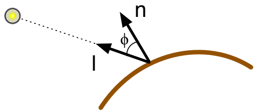
Objetivo

- Determinar a quantidade de luz irradiada do ponto em direção à câmera

Modelo de iluminação de Phong

- Componente difusa

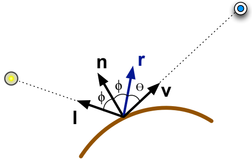
$$\mathbf{c} = \mathbf{m}_{dif} \ \max(0, \hat{\mathbf{n}} \cdot \hat{\mathbf{l}})$$



- Componente especular (*glossy*)
  - Se existir componente difusa

$$\mathbf{c} \mathrel{+}= \mathbf{m}_{spe} \ \max(0, \hat{\mathbf{r}} \cdot \hat{\mathbf{v}})^{shi}$$

onde: $\vec{\mathbf{r}} = 2(\vec{\mathbf{n}} \cdot \vec{\mathbf{l}})\vec{\mathbf{n}} - \vec{\mathbf{l}}$ (vetor reflexão)



Composição da cena

Instâncias de objetos

- Objetos que recebem iluminação
  - Representação: forma geométrica, material
- Objetos que são fontes de luz
  - Representação: forma geométrica, fonte de luz

Composição da cena

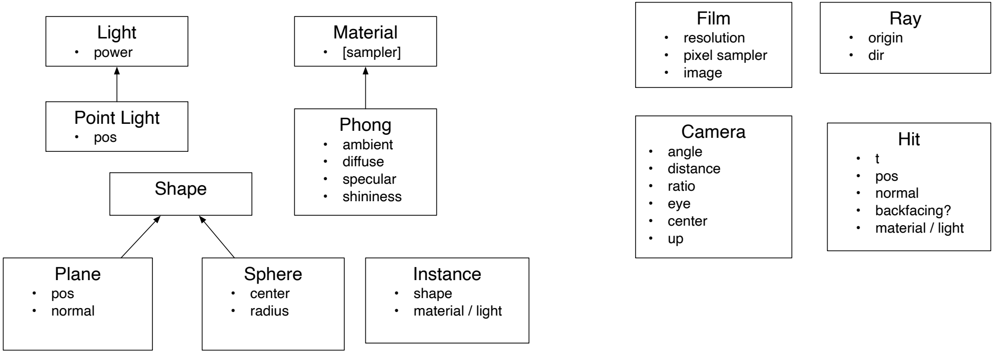
Instâncias de objetos

- Objetos que recebem iluminação
  - Representação: forma geométrica, material
- Objetos que são fontes de luz
  - Representação: forma geométrica, fonte de luz

Ponto de interseção (*Hit*)

- Representação: $\mathbf{p}$, $\hat{\mathbf{n}}$, *backfacing floag*, instance

Classes de objetos



Traçado de raio

Algoritmo de traçado de raios

$$
\begin{aligned}
&Render(film, camera, scene)\\
&\quad \textbf{for}\ \ each\ pixel(i,j)\ in\ film\\
&\qquad x_n, y_n = film.GetSample(i,j)\\
&\qquad ray = camera.GenerateRay(x_n, y_n)\\
&\qquad \mathbf{c} = scene.TraceRay(ray)\\
&\quad film.SetValue(i,j,\mathbf{c})
\end{aligned}
$$

Traçado de raio

Algoritmo de traçado de raios

$Render(film, camera, scene)$
for $each\ pixel(i,j)\ in\ film$
$x_n, y_n = film.GetSample(i,j)$
$ray = camera.GenerateRay(x_n, y_n)$
$\mathbf{c} = scene.TraceRay(ray)$
$film.SetValue(i,j,\mathbf{c})$

$Scene.TraceRay(ray)$
$hit = ComputeIntersection(ray)$
if hit **then**
if $hit.light$ **then**
$r = hit.t$
$\mathbf{c} = hit.light.P/r^2$
else
$\mathbf{c} = hit.material.Eval(this, hit, ray.\mathbf{o})$
return c

Traçado de raio

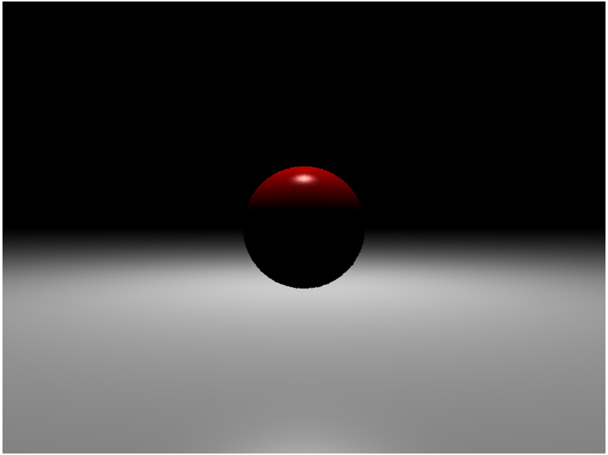
Cálculo de radiância

$PhongMaterial.Eval(scene, hit, \mathbf{o})$
$\quad \mathbf{c} = (0, 0, 0)$
$\quad \hat{\mathbf{v}} = normalize(\mathbf{o} - hit.\mathbf{p})$
$\quad$ **for** $each\ light\ source\ l_s\ in\ scene$
$\qquad \mathbf{L}_i, \hat{\mathbf{l}} = l_s.Radiance(scene, hit.\mathbf{p})$
$\qquad \mathbf{c} \mathrel{+}= \mathbf{m}_{dif} * \mathbf{L}_i * \hat{\mathbf{n}} \cdot \hat{\mathbf{l}}$
$\qquad \hat{\mathbf{r}} = reflect(-\hat{\mathbf{l}}, \hat{\mathbf{n}})$
$\qquad \mathbf{c} \mathrel{+}= \mathbf{m}_{spe} * \max(0, \hat{\mathbf{r}} \cdot \hat{\mathbf{v}})^{shi}$
$\quad$ **return c**

$Light.Radiance(scene, \mathbf{p})$
$\quad \hat{\mathbf{l}} = normalize(this.\mathbf{p} - \mathbf{p})$
$\quad r = distance(\mathbf{p}, this.\mathbf{p})$
$\quad \mathbf{L}_i = this.P/r^2$
$\quad$ **return** $\mathbf{L}_i, \hat{\mathbf{l}}$

Traçado de raio

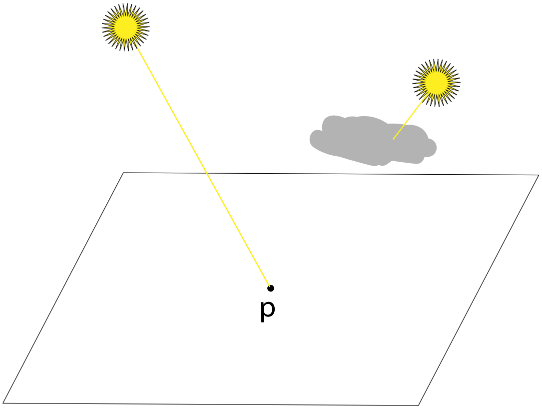
Versão 0: modelo de iluminação de Phong



Geração de sombras

Visibilidade das fontes de luz

- Traçado de raio auxiliar para teste de visibilidade



Imagens extraídas de Physically Based Rendering, Pharr et al. (www.pbr-book.org)

Geração de sombras

$Light.Radiance(scene, \mathbf{p})$

$\hat{\mathbf{l}} = normalize(this.\mathbf{p} - \mathbf{p})$

$ray = Ray(hit.\mathbf{p}, \hat{\mathbf{l}})$

$hit_s = scene.ComputeIntersection(ray)$

if $hit_s.light == this$ **then**

$r = distance(this.\mathbf{p}, hit.\mathbf{p})$

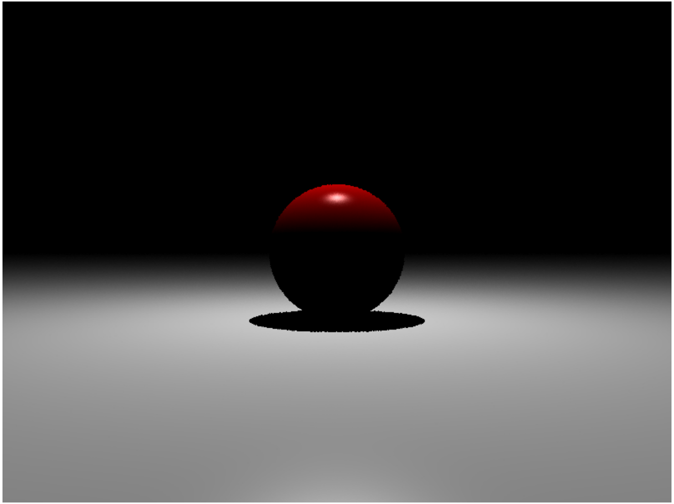
$\mathbf{L}_i = this.P/r^2$

return $\mathbf{L}_i, \hat{\mathbf{l}}$

else

return $(0, 0, 0), (0, 0, 0)$

Geração de sombras

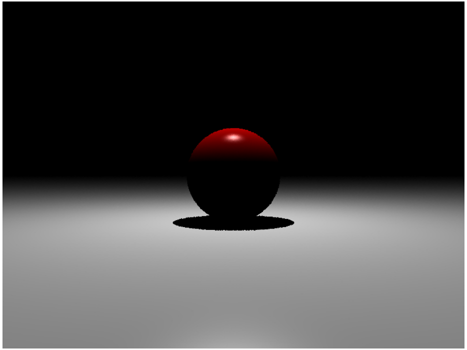


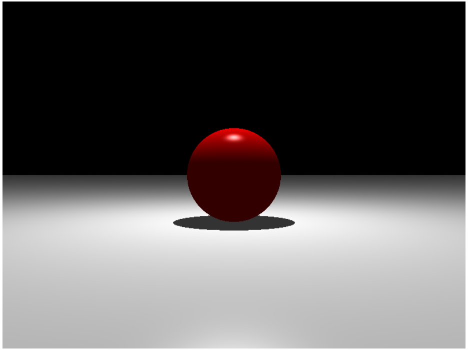
Luz ambiente

Emulação de luz indireta

- Valor global, associado à cena, independente de fonte de luz: $\mathbf{l}_{amb}$
- Material passa a ter componente ambiente: $\mathbf{m}_{amb}$

$$\mathbf{c} = \mathbf{m}_{amb} * \mathbf{l}_{amb}$$





Luz ambiente

- Incluindo no traçado de raio

$Scene.TraceRay(ray)$
$\quad hit = ComputeIntersection(ray)$
$\quad$ **if** hit **then**
$\quad\quad$ **if** $hit.light$ **then**
$\quad\quad\quad r = hit.t$
$\quad\quad\quad \mathbf{c} = \mathbf{l}_{amb} + hit.light.P/r^2$
$\quad\quad$ **else**
$\quad\quad\quad \mathbf{c} \mathrel{+}= hit.material.Eval(this, hit, ray.\mathbf{o})$
$\quad\quad$ **return c**

$PhongMaterial.Eval(scene, hit, \mathbf{o})$
$\quad \mathbf{c} = \mathbf{m}_{amb} * scene.GetAmbientLight()$
$\quad \hat{\mathbf{v}} = normalize(\mathbf{o} - hit.\mathbf{p})$
$\quad$ **for** $each\ light\ source\ l_s\ in\ scene$
$\quad\quad \mathbf{L}_i, \hat{\mathbf{l}} = l_s.Radiance(scene, hit.\mathbf{p})$
$\quad\quad \mathbf{c} \mathrel{+}= \mathbf{m}_{dif} * \mathbf{L}_i * \hat{\mathbf{n}} \cdot \hat{\mathbf{l}}$
$\quad\quad \hat{\mathbf{r}} = reflect(-\hat{\mathbf{l}}, \hat{\mathbf{n}})$
$\quad\quad \mathbf{c} \mathrel{+}= \mathbf{m}_{spe} * \max(0, \hat{\mathbf{r}} \cdot \hat{\mathbf{v}})^{shi}$
$\quad$ **return c**