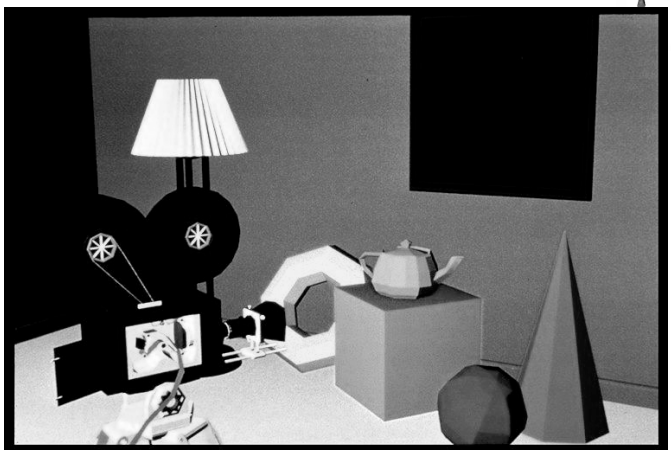
Assunto última aula

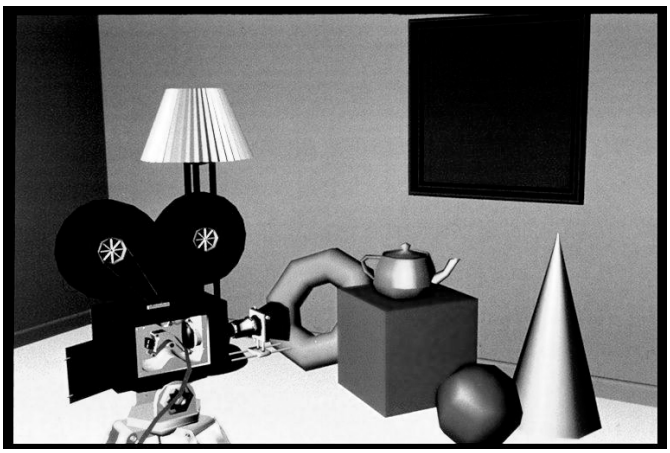
- Modelos de Iluminação para Sombreamento de Polígonos

Flat Shading

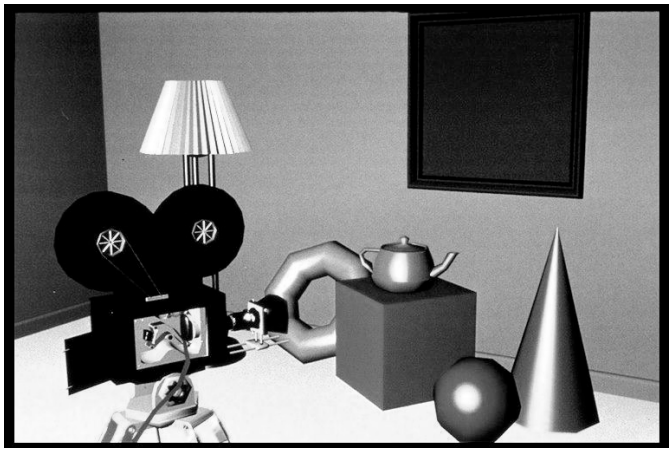


Pixar Shutterbug sequence

Gouraud Shading



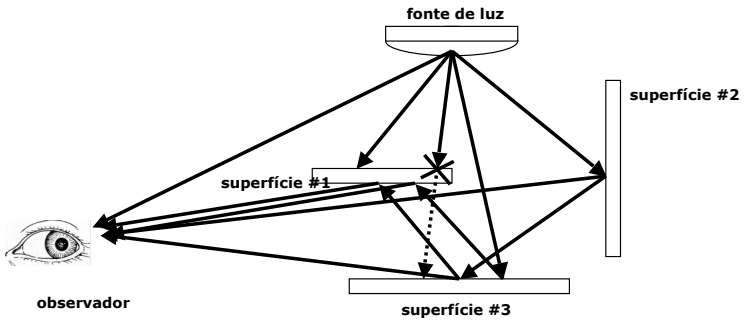
Phong Shading



Iluminação Local

- O cálculo de iluminação num ponto da superfície independe da energia recebida *indiretamente*
- Toda informação necessária para este cálculo é LOCAL
- Parcela Ambiente simula este efeito

Síntese de Imagens Realísticas

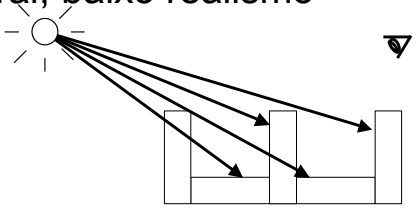


Modelos de Iluminação

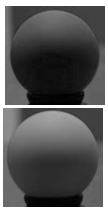
- Descrevem como a luz
  - Interage com os materiais
  - É transportada na cena (light transport)
  - Atinge o observador
- Categorias
  - Modelos de Iluminação Locais
  - Modelos de Iluminação Globais

Modelos de Iluminação Locais

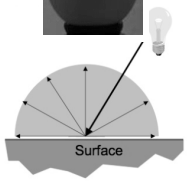
- Não consideram inter-reflexões
- Rápidos para cálculo
- Não são fisicamente corretos
- Em geral, baixo realismo

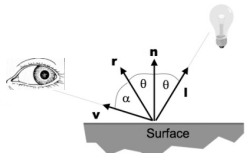


Exemplo Phong



+





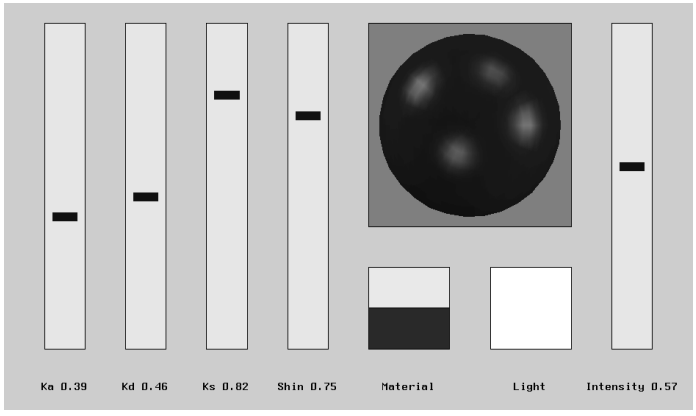
$$I = \boxed{I_a k_a} + \Sigma\{I_m [\boxed{k_d (N.L)} + \boxed{k_s (R.V)^q}]\}$$

Ambiente Difusa Especular

9

Marcelo Walter - UFPE

Especificação Usual dos Materiais

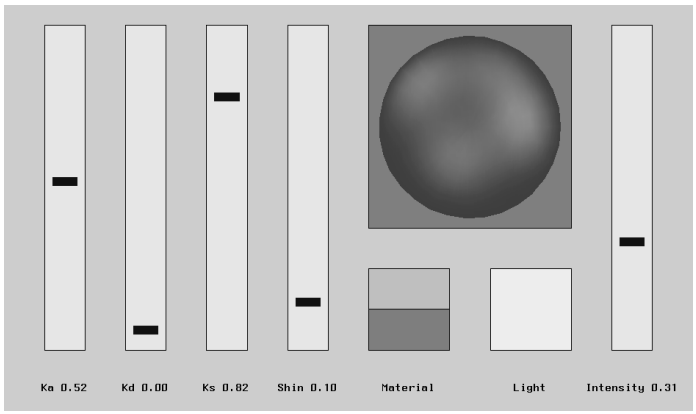


Plástico

10

Marcelo Walter - UFPE

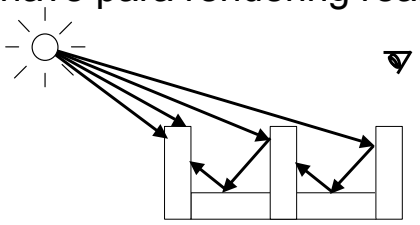
Especificação Usual dos Materiais

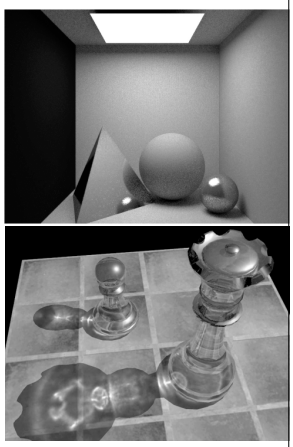


Cobre

Modelos de Iluminação Globais

- Toda a cena é considerada
- Consideram inter-reflexões
- Maior custo computacional
- Chave para rendering realista

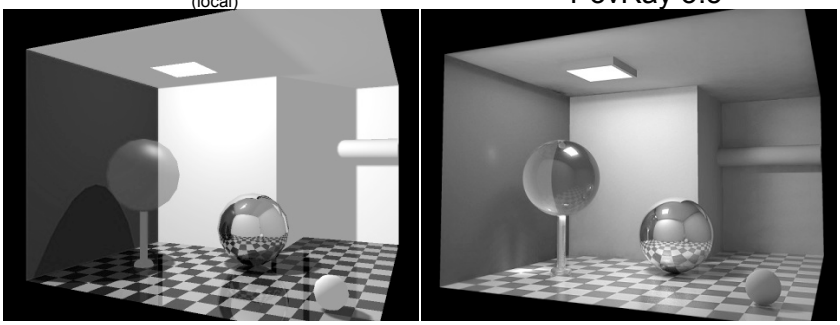




Comparando…

OpenGL
(local)

PovRay 3.5

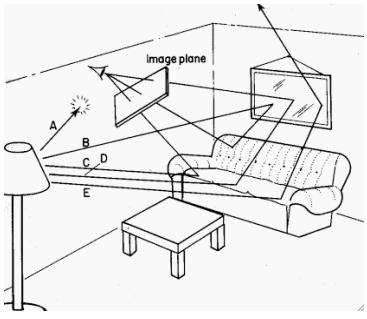


http://www.winosi.onlinehome.de/Gallery_t14_01.htm

Ray Casting

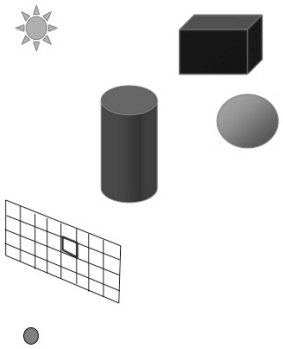
(ou de onde saiu a idéia de raios em primeiro lugar!)

```
Para cada pixel da tela
   Construa um raio a partir do olho
   Para cada objeto na cena
      Encontre a intersecção com o raio
      Mantenha se for a mais próxima
   Calcule a iluminação neste ponto
```

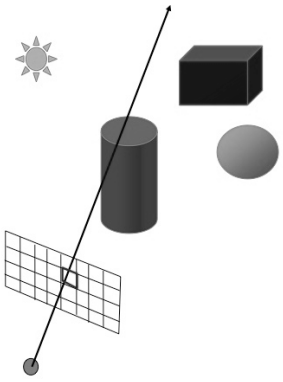


Arthur Appel. *Some Techniques for Shading Machine Renderings of Solids*
AFIPS 1968 Spring Joint Computer Conf, p. 37-45, **1968!!**

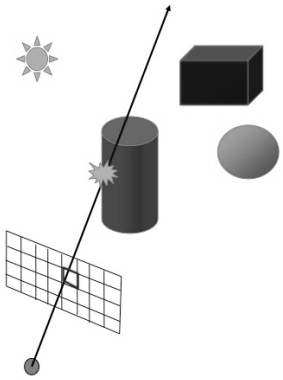
Ray Casting



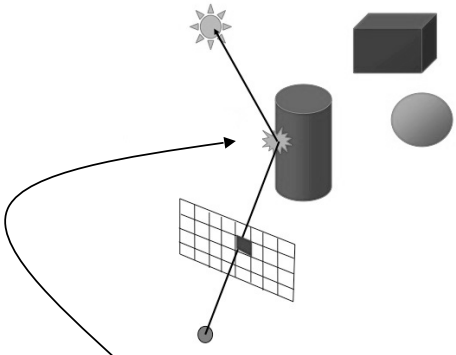
Ray Casting



Ray Casting

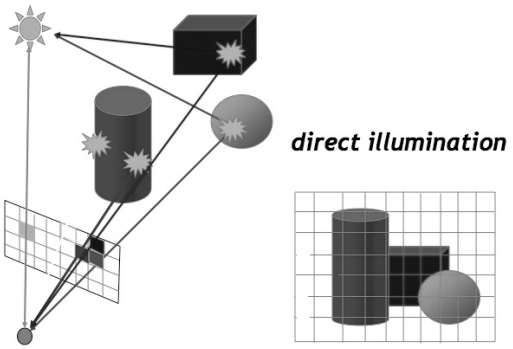


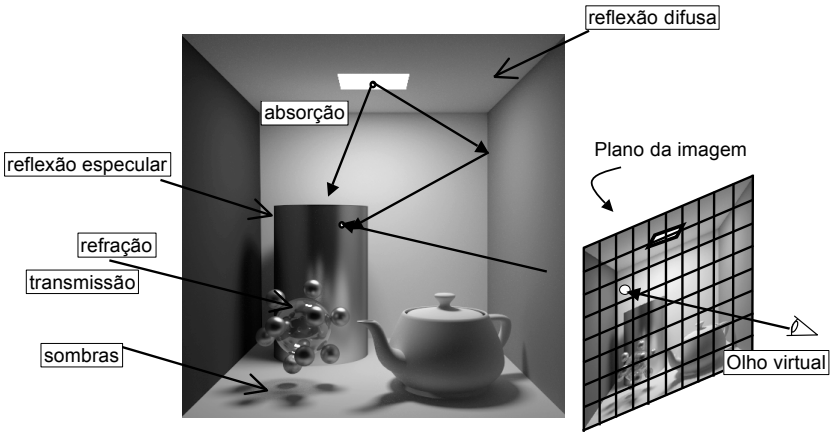
Ray Casting



$$I = I_a k_a + \Sigma \{ I_{pm} [k_d (N.L) + k_s (R.V)^q] \}$$

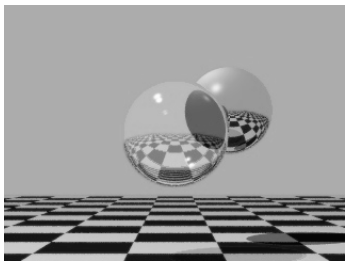
Ray Casting



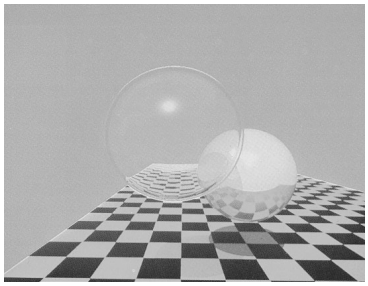


Principais fenômenos que podem acontecer na interação entre luz e objetos

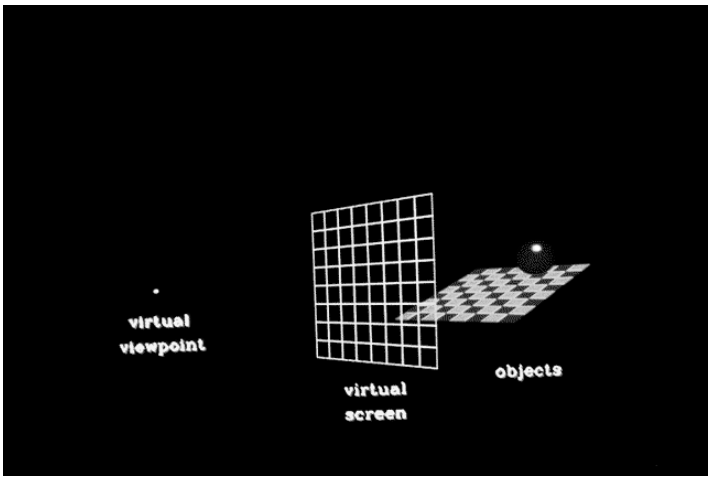
Traçado de Raios



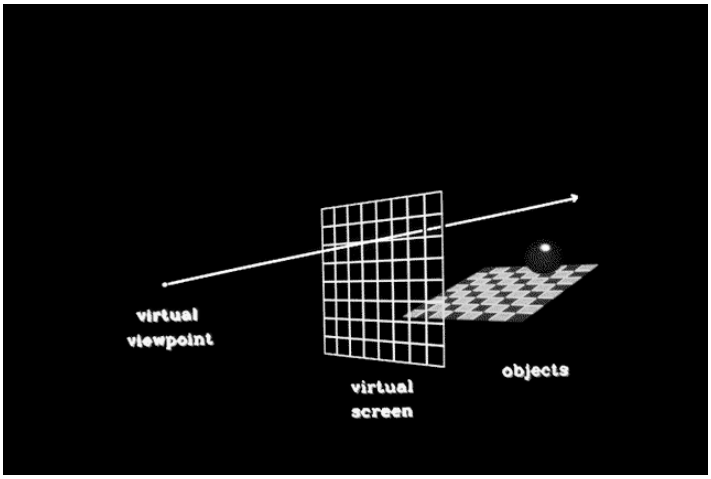
- Primeiras idéias em 1968 (*Ray Casting*)
- 1980 Turner Whitted
  - *Communications of the ACM* N. 23, V. 6, June 1980, p. 343-349)



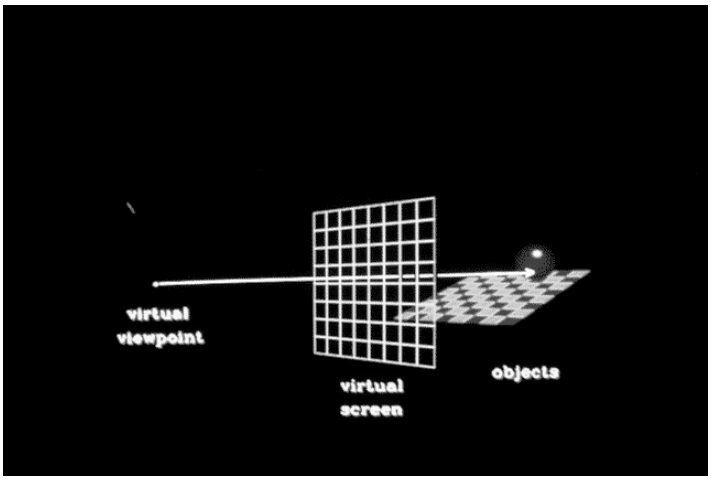
Traçado de Raios – Situação Inicial



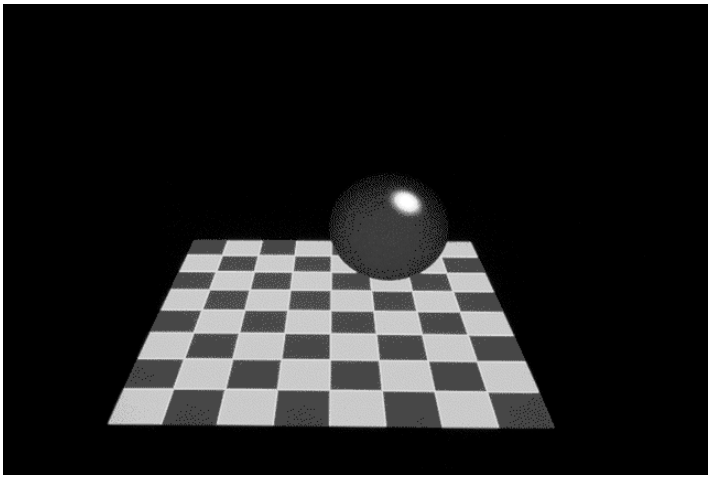
Traçado de Raios – Disparando Raios



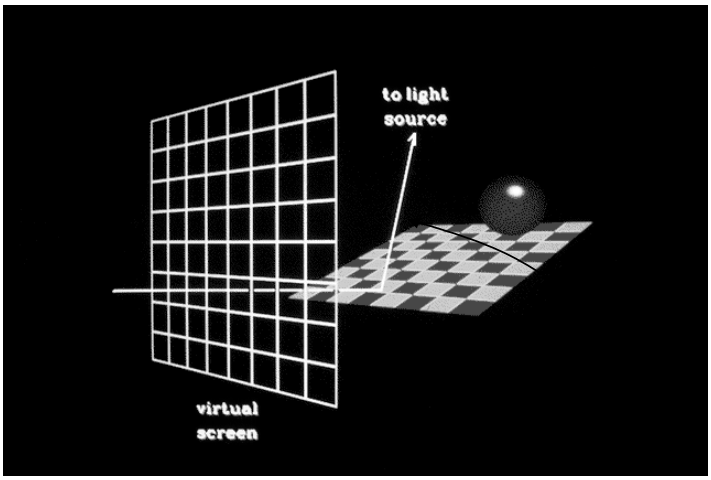
Traçado de Raios – Disparando Raios



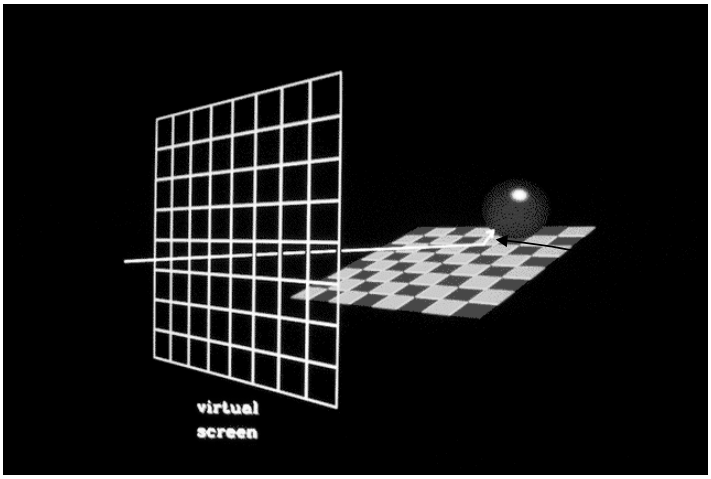
Traçado de Raios – Resultado 1



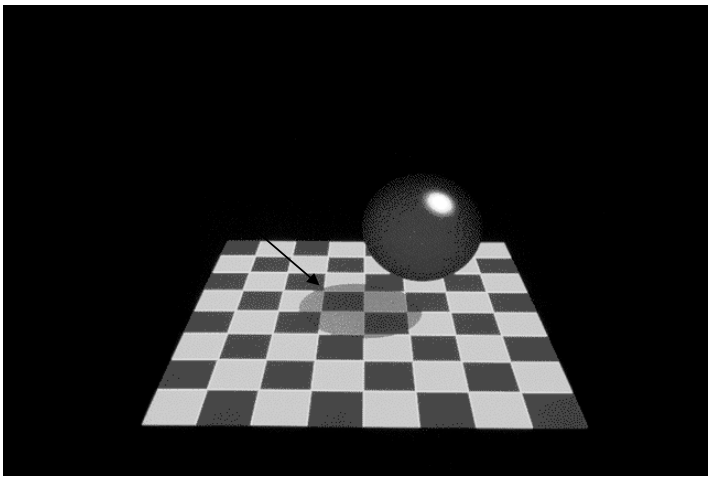
Traçado de Raios – Raio de Sombra



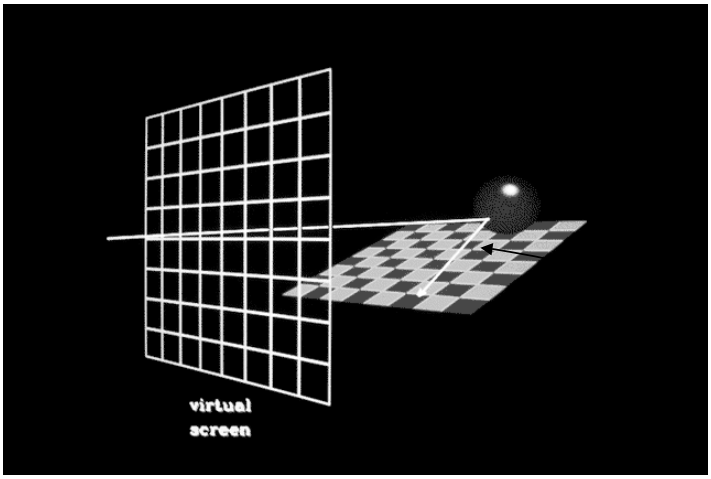
Traçado de Raios – Raio de Sombra



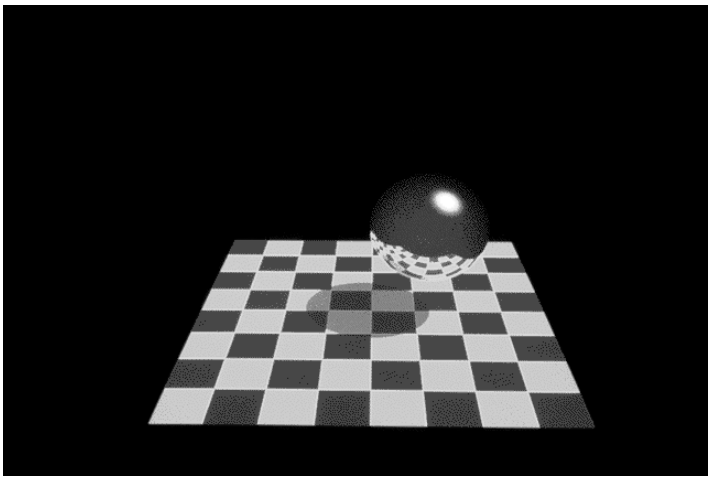
Traçado de Raios – Resultado 2



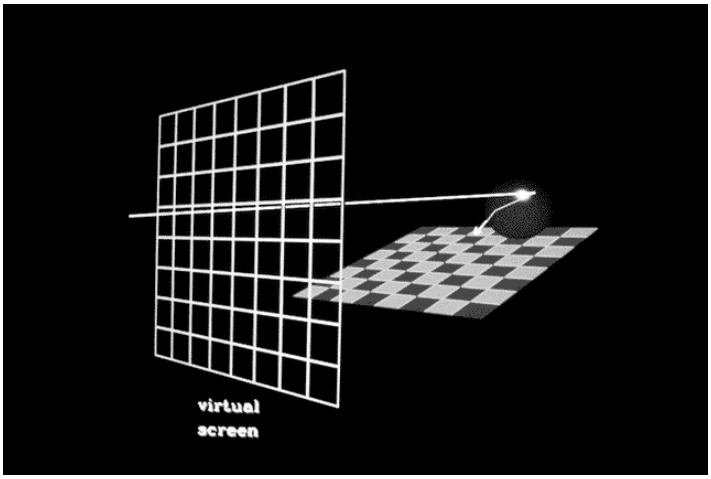
Traçado de Raios – Raio de Reflexão



Traçado de Raios – Resultado 3



Traçado de Raios – Raio Transmitido



Traçado de Raios

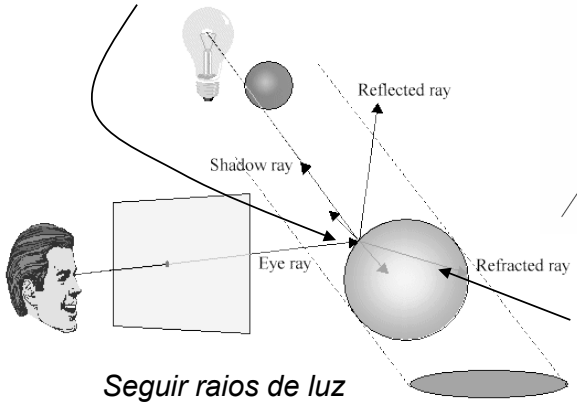
$$I = I_{local} + k_{rg} I_{reflexão} + k_{tg} I_{transmitida}$$

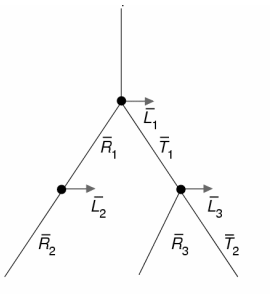
$$I_{local} = I_a k_a + I_p [k_d (N.L) + k_r (R.V)^n]$$

Modelo Iluminação Simples da última aula

Árvore de Raios

$$I = I_{local} + k_{rg} I_{reflexão} + k_{tg} I_{transmitida}$$



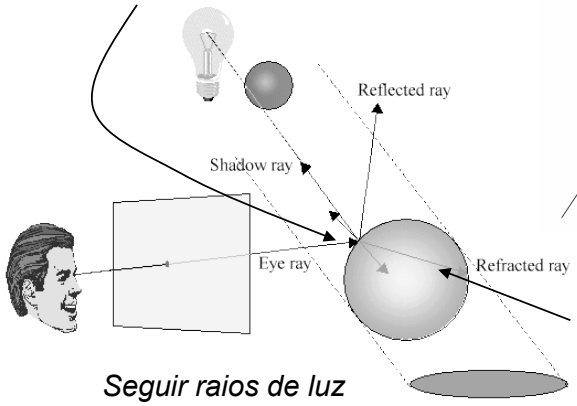


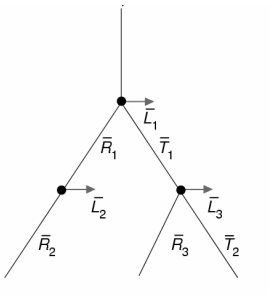
Também conhecido como raio transmitido

Seguir raios de luz no ambiente, a partir do observador, recursivamente

Árvore de Raios

$$I = I_{local} + k_{rg} I_{reflexão} + k_{tg} I_{transmitida}$$





Também conhecido como raio transmitido

Seguir raios de luz no ambiente, a partir do observador, recursivamente

Pseudocódigo (aqui falta um *loop* para incluir vários objetos da cena e um *loop* para as fontes de luz)

```
void RT( Point3D start, Point3D end, int depth, RGB *color)
{
if ( depth > MAXDEPTH ) *color = BLACK;
else {/* verifica se raio intersecta algum objeto. Caso positivo retorna o mais
próximo */
  if ( rayHit ( start, end, &hitObject, &hitPoint )){

     /* contribuicao local */
     shade( hitObject, hitPoint, localColor );

     /* calcula direções de reflexão e transmissão */
     calcReflection( hitObject, hitPoint, &reflectDirection);
     calcTrans( hitObject, hitPoint, &transmDirection );

     /* Chamadas recursivas */
     RT ( hitPoint, reflectDirection, depth+1, &reflectedColor );
     RT ( hitPoint, transmDirection, depth+1, &transmColor );

     /* combina cores */
     combineColor( hitObject, localColour, reflectedColour, transmColour, color);
  }
  else *color = BLACK;
}
```


E as sombras?

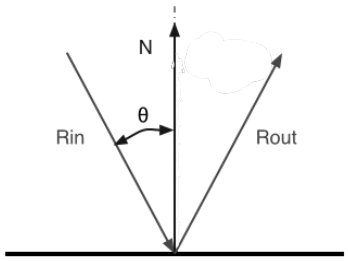
```
void RT( Point3D start, Point3D end, int depth, RGB *color)
{
if ( depth > MAXDEPTH ) *color = BLACK;
else {/* verifica se raio intersecta algum objeto. Caso positivo retorna o mais
próximo */
  if ( rayHit ( start, end, &hitObject, &hitPoint )){

     /* contribuicao local */
     shade( hitObject, hitPoint, localColor );

     /* calcula direções de reflexão e transmissão */
     calcReflection( hitObject, hitPoint, &reflectDirection);
     calcTrans( hitObject, hitPoint, &transmDirection );

     /* Chamadas recursivas */
     RT ( hitPoint, reflectDirection, depth+1, &reflectedColor );
     RT ( hitPoint, transmDirection, depth+1, &transmColor );

     /* combina cores */
     combineColor( hitObject, localColour, reflectedColour, transmColour, color);
  }
  else *color = BLACK;
}
```

Aqui verifica se a luz atinge ou não o ponto

Cálculo do Vetor Refletido

```
void RT( Point3D start, Point3D end, int depth, RGB *color)
{
if ( depth > MAXDEPTH ) *color = BLACK;
else {/* verifica se raio intersecta algum objeto. Caso positivo retorna o mais
próximo */
  if ( rayHit ( start, end, &hitObject, &hitPoint )){

     /* contribuicao local */
     shade( hitObject, hitPoint, localColor );

     /* calcula direções de reflexão e transmissão */
     calcReflection( hitObject, hitPoint, &reflectDirection);
     calcTrans( hitObject, hitPoint, &transmDirection );

     /* Chamadas recursivas */
     RT ( hitPoint, reflectDirection, depth+1, &reflectedColor );
     RT ( hitPoint, transmDirection, depth+1, &transmColor );

     /* combina cores */
     combineColor( hitObject, localColour, reflectedColour, transmColour, color);
  }
  else *color = BLACK;
}
```


Cálculo do Vetor Refletido

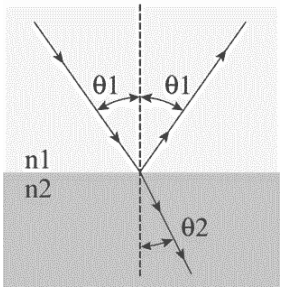


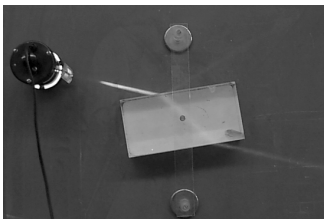
$$R_{out} = 2N\,(N.R_{in}) - R_{in}$$

Cálculo do Vetor Transmitido

```
void RT( Point3D start, Point3D end, int depth, RGB *color)
{
if ( depth > MAXDEPTH ) *color = BLACK;
else {/* verifica se raio intersecta algum objeto. Caso positivo retorna o mais
próximo */
  if ( rayHit ( start, end, &hitObject, &hitPoint )){

     /* contribuicao local */
     shade( hitObject, hitPoint, localColor );

     /* calcula direções de reflexão e transmissão */
     calcReflection( hitObject, hitPoint, &reflectDirection);
     calcTrans( hitObject, hitPoint, &transmDirection );

     /* Chamadas recursivas */
     RT ( hitPoint, reflectDirection, depth+1, &reflectedColor );
     RT ( hitPoint, transmDirection, depth+1, &transmColor );

     /* combina cores */
     combineColor( hitObject, localColour, reflectedColour, transmColour, color);
  }
  else *color = BLACK;
}
```


Cálculo do Vetor Transmitido





Alguns índices de refração

ar = 1
água a 20ºC = 1.33
Gelo=1.31
vidro=1.5
diamante=2.417

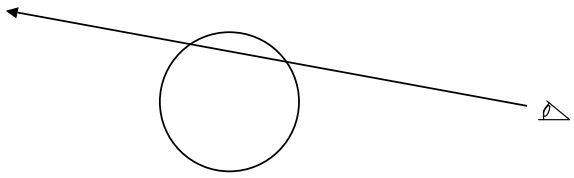
Lei de Snell
n2 sen θ_1 = n1 sen θ_2

Cálculo de Intersecções

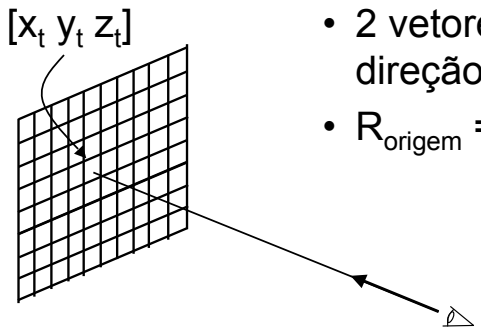
```
void RT( Point3D start, Point3D end, int depth, RGB *color)
{
if ( depth > MAXDEPTH ) *color = BLACK;
else {/* verifica se raio intersecta algum objeto. Caso positivo retorna o mais
próximo */
  if ( rayHit ( start, end, &hitObject, &hitPoint ){

     /* contribuicao local */
     shade( hitObject, hitPoint, localColor );

     /* calcula direções de reflexão e transmissão */
     calcReflection( hitObject, hitPoint, &reflectDirection);
     calcTrans( hitObject, hitPoint, &transmDirection );

     /* Chamadas recursivas */
     RT ( hitPoint, reflectDirection, depth+1, &reflectedColor );
     RT ( hitPoint, transmDirection, depth+1, &transmColor );

     /* combina cores */
     combineColor( hitObject, localColour, reflectedColour, transmColour, color);
  }
  else *color = BLACK;
}
```


Cálculo das Intersecções

- Gargalo (até 95% do tempo total)
- Casos mais simples resolvem de forma analítica
- Raio/esfera - resolução de uma eq. de 2o. grau



Preliminares



- 2 vetores: origem e direção
- $R_{origem} = R_0 = [x_0\ y_0\ z_0]$

$R_{direção} = R_d = [x_d\ y_d\ z_d] =$
$[(x_t - x_0)\ (y_t - y_0)\ (z_t - z_0)]$

Preliminares

- Definição paramétrica do raio
- $R(t) = R_0 + R_d \cdot t$
  - R_d deve ser normalizado
- t = 0 estamos na origem
- t = 1 estamos na tela
- t > 1 onde há objetos

Preliminares

- Definição paramétrica da esfera
- $(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = r^2$
- Centro da esfera = $[x_c\ y_c\ z_c]$
- r = raio da esfera

Encontrando a intersecção

- Substituindo a eq. paramétrica do raio na eq. da esfera temos:

$$(x_0 + x_d t - x_c)^2 + (y_0 + y_d t - y_c)^2 + (z_0 + z_d t - z_c)^2 = r^2$$

Desenvolvendo esta equação ficamos com uma eq. do 2º grau em t

- $A\ t^2 + B\ t + C = 0$
  onde
  $A = (x_d^2 + y_d^2 + z_d^2) = 1$ (pq?)
  $B = 2\ (x_d(x_0 - x_c) + y_d(y_0 - y_c) + z_d(z_0 - z_c))$
  $C = (x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2 - r^2$

- Resolução por Baskara
  - Determinante: $\Delta = B^2 - 4AC$
  - Se $\Delta < 0$ o raio não atinge a esfera
  - Se $\Delta = 0$ o raio tangencia a esfera
  - Caso contrário encontramos as raízes t_0 e t_1. A menor raiz positiva corresponde ao ponto de intersecção mais próximo
    - t_0 = -B - sqrt(Δ) / 2
    - t1 = -B + sqrt(Δ) / 2

Se t_0 for positivo não precisamos calcular t_1. Porque?

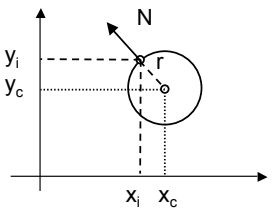
Encontrando o ponto de intersecção

o menor de t0 ou t1

$$Pi = [x_0 + x_d\ t \quad y_0 + y_d\ t \quad z_0 + z_d\ t]$$

E o vetor normal?

$$N = [(x_i - x_c)/r \quad (y_i - y_c)/r \quad (z_i - z_c)/r\ ]$$



Superfícies Quádricas

- Esfera é um caso especial de quádrica
- Fórmula genérica

$$ax^2 + by^2 + cz^2 + 2dxy + 2eyz + 2fxz + 2gx + 2hy + 2jz + k = 0$$

- 10 coeficientes (a,b,c,…,k)
- Exemplo: esfera de raio=3 centrada em (2,3,-4)

Superfícies Quádricas

$$(x-2)^2 + (y-3)^2 + (z+4)^2 = 9$$

$$x^2 - 4x + 4 + y^2 - 6y + 9 + z^2 + 8z + 16 - 9 = 0$$

$$x^2 + y^2 + z^2 - 4x - 6y + 8z + 20 = 0$$

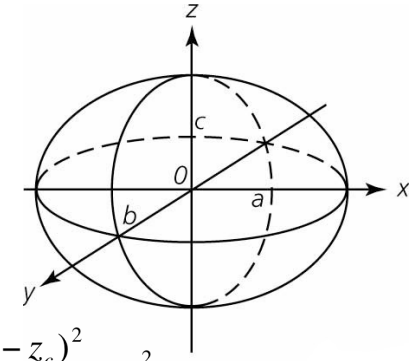
$$ax^2 + by^2 + cz^2 + 2dxy + 2eyz + 2fxz + 2gx + 2hy + 2jz + k = 0$$

- a=b=c=1
- d=e=f=0
- g=-2, h=-3, j=4, k=20

Especifica inequivocamente a esfera

Outras Quádricas

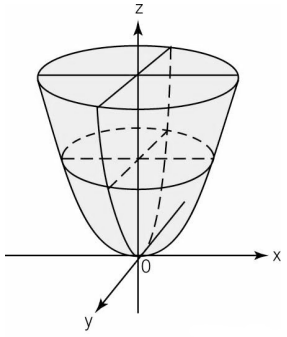
- Elipsóide



$$\frac{(x-x_c)^2}{a^2}+\frac{(y-y_c)^2}{b^2}+\frac{(z-z_c)^2}{c^2}=r^2$$

Outras Quádricas

- Parabolóide



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

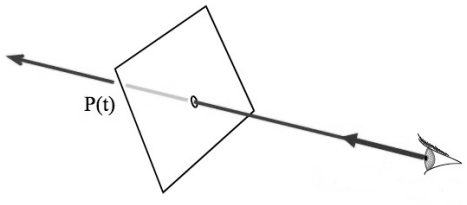
Outras Quádricas

- Cilindro Infinito

$$(x - x_c)^2 + (y - y_c)^2 = R_2^2$$

Outras Quádricas - Plano

- Caso Especial
- Eq do plano: $ax + by + cz + d = 0$
- Eq do raio: $P(t) = (x_0 + x_d t, y_0 + y_d t, z_0 + z_d t)$



Outras Quádricas - Plano

- Substituindo:

$$a(x_0 + x_d t) + b(y_0 + y_d t) + c(z_0 + z_d t) + d = 0$$

$$t(ax_d + by_d + cz_d) = -ax_0 - by_0 - cz_0 - d$$

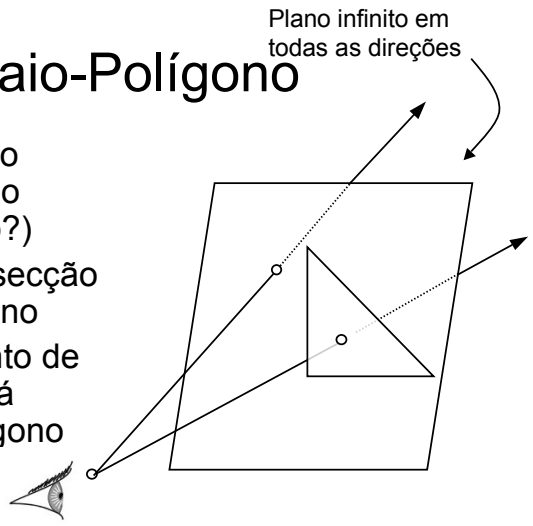
$$t = -\frac{ax_0 + by_0 + cz_0 + d}{(ax_d + by_d + cz_d)}$$

Como sabemos se o raio e plano não são paralelos entre si?

E se t for negativo?

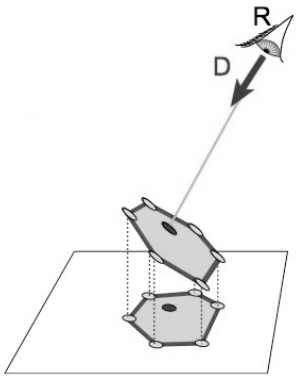
Raio-Polígono

- Encontrar eq. do plano suporte ao polígono (como?)
- Encontrar intersecção do raio com plano
- Verificar se ponto de intersecção está contido no polígono



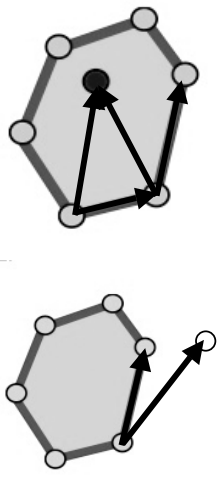
Teste de contenção ponto-polígono

- Projeta o polígono em 2D (despreza uma dimensão)
- Verifica o sinal do produto vetorial entre os pares de vetores formados pelo ponto e os vértices



Teste de contenção ponto-polígono

- Verifica o sinal da coord. z do produto vetorial entre os pares de vetores formados pelo ponto e os vértices. Todos devem ter o mesmo sinal para o ponto estar contido

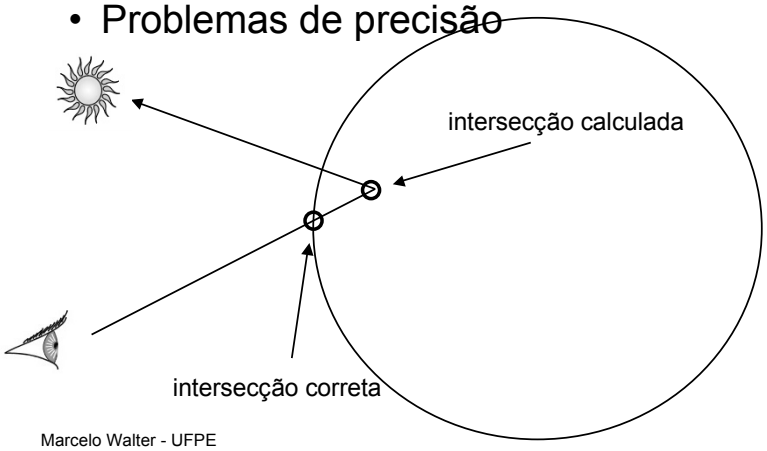


Dicas de Implementação

- Otimização do raio de sombra
  - Diferentemente do raio do olho, não precisamos encontrar TODAS as intersecções e manter a mais próxima. Precisamos apenas definir se HÁ ou NÃO HÁ intersecção
  - Pára (do verbo parar) o laço de teste na primeira intersecção

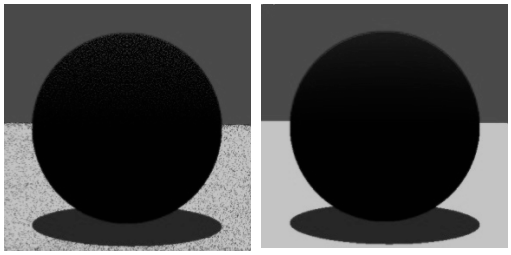
Dicas de Implementação

- Problemas de precisão



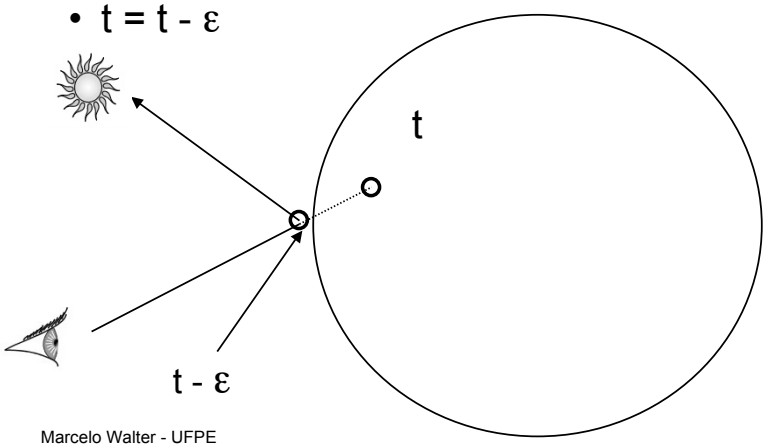
Dicas de Implementação

- A própria superfície se bloqueia
- Solução: incorporar um valor pequeno que é diminuído de t para remover o ponto de dentro da esfera



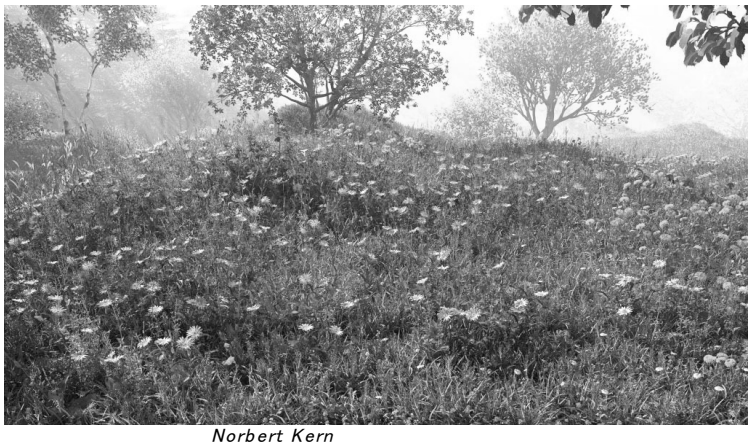
Dicas de Implementação

- $t = t - \varepsilon$



Galeria de Imagens

Internet Ray Tracing Competition



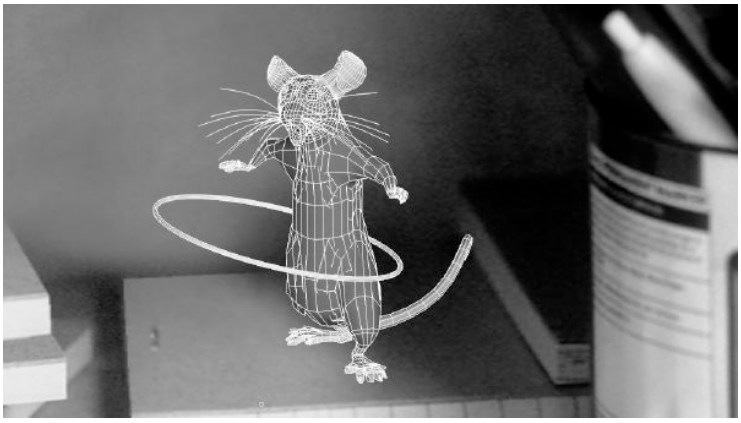
Norbert Kern
RENDERER USED: Megapov
RENDER TIME: 4 semanas! (4096x2304)
HARDWARE USED: P4 3.06 GHz / 2 GB RAM

IRTC

Roy Rodriguez
RENDERER USED:
Povray 3.6
RENDER TIME: 11h 24min
HARDWARE USED:
Turbo 3400+X AthlonXP 2200+

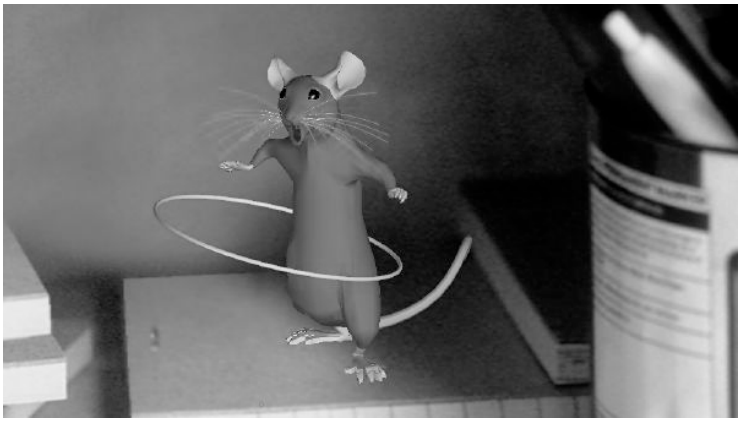
65

Galeria – Mental Ray

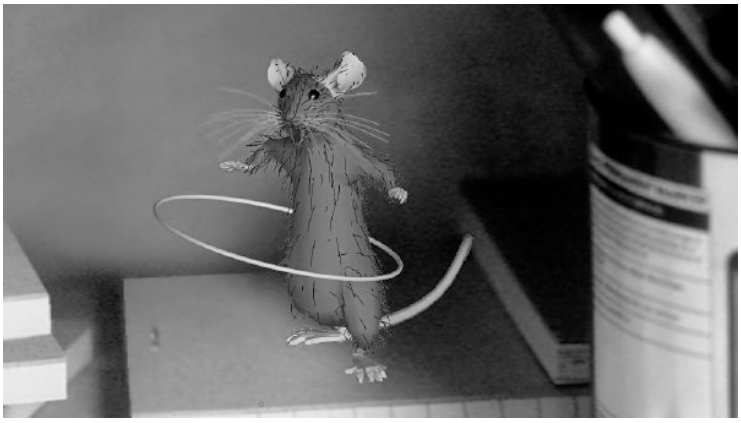


66

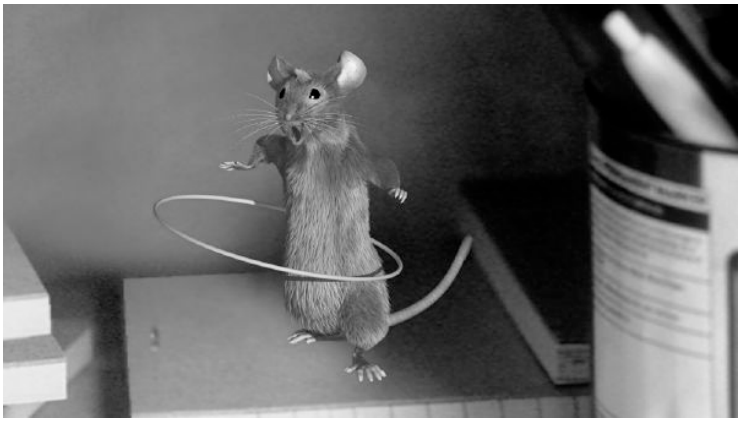
Galeria – Mental Ray



Galeria – Mental Ray



Galeria – Mental Ray



Galeria – Mental Ray

Maiores Referências

- An Introduction to Ray Tracing - A. Glassner Academic Press, 1989
- Internet Ray Tracing Competition
  www.irtc.org
- Softwares
  - http://www.povray.org/
  - Rayshade