Mestrado em
Engenharia Informática

Ray Tracing Clássico

Visualização e Iluminação

Luís Paulo Peixoto dos Santos

Introdução

- O conceito de *ray tracing* foi introduzido na Computação Gráfica em 1980 por Whitted
- O termo *ray tracing* é usado livremente para designar uma infinidade de diferentes abordagens ao problema de *rendering,* desde que baseados no princípio de intersecção de uma semirecta (raio) com primitivas geométricas
- Este operador básico, frequentemente designado por *ray casting*, permite determinar a **visibilidade** a partir de um ponto e ao longo de uma direcção
- Esta sessão debruça-se sobre o algoritmo clássico de *ray tracing* conforme apresentado por Whitted, logo **determinístico**

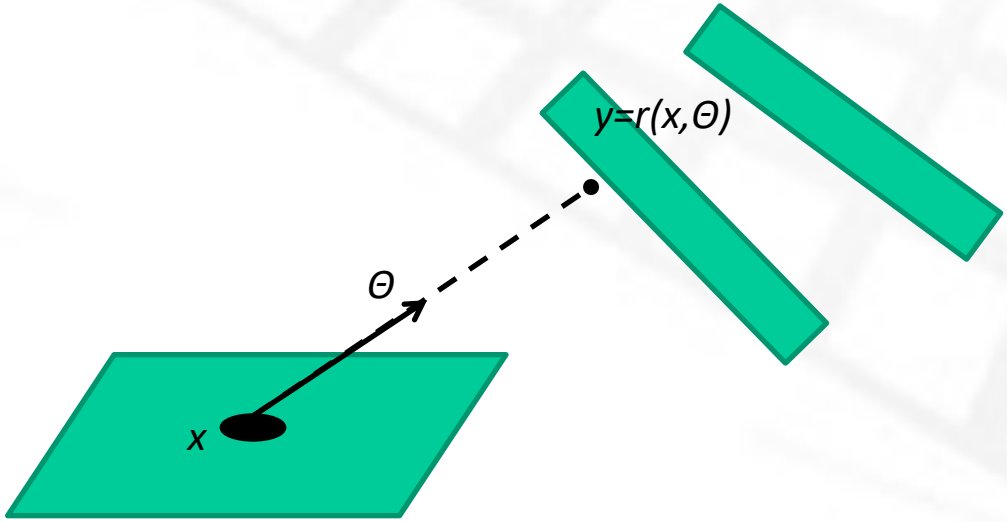
Ray tracing: definição

- *Ray tracing*: $y = r(x, \Theta)$
  y é o ponto visível ao longo de um raio com origem em *x* e direcção *Θ*.

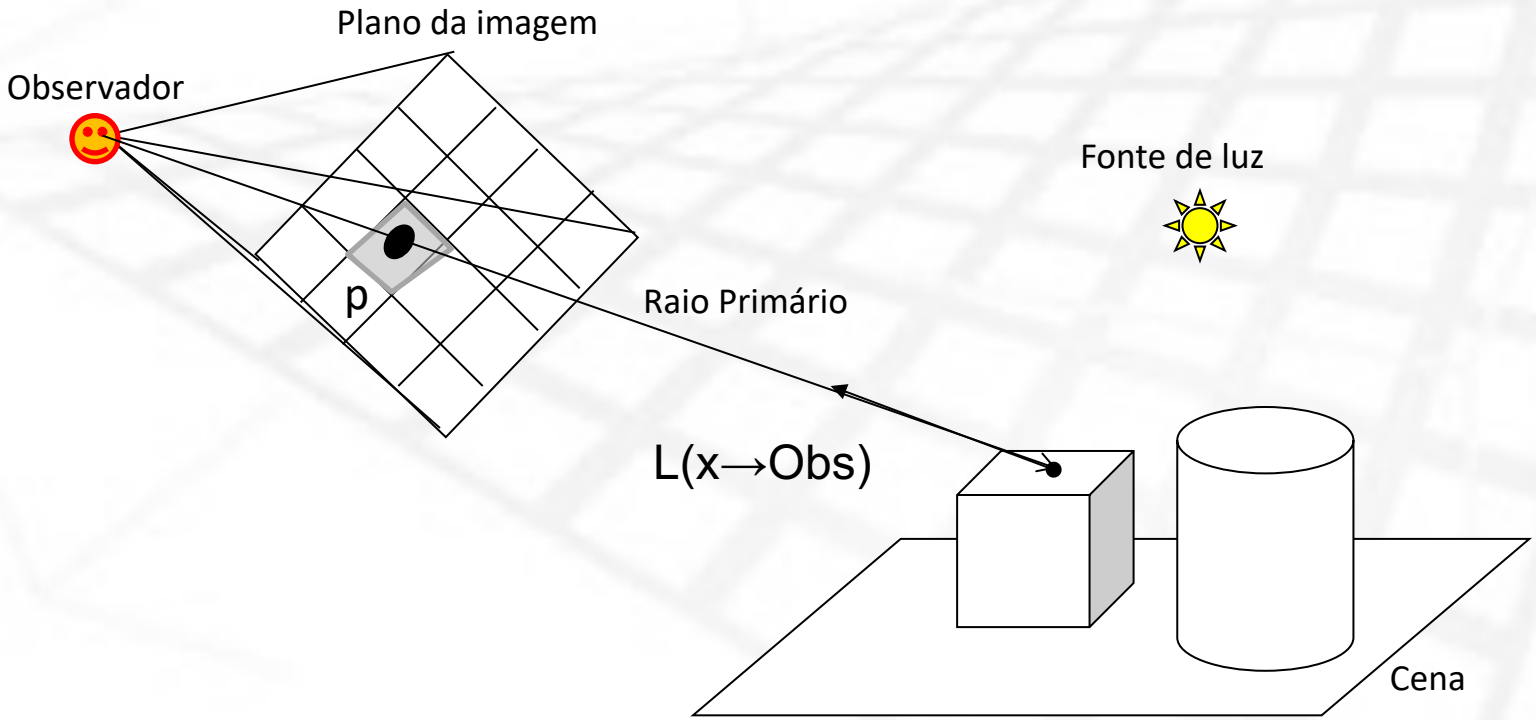
$$r(x,\Theta) = \{y : y = x + \Theta t_{\text{intersection}}\}$$

$$t_{\text{intersection}} = \min\{t : t > 0 \wedge x + \Theta t \in A\}$$

sendo *A* o conjunto de todos os objectos representados na cena.



Ray Tracing: Princípios



Raios primários determinam a **visibilidade directa**: quais os objectos directamente acessíveis ao sensor.

Ray Tracing: Algoritmo

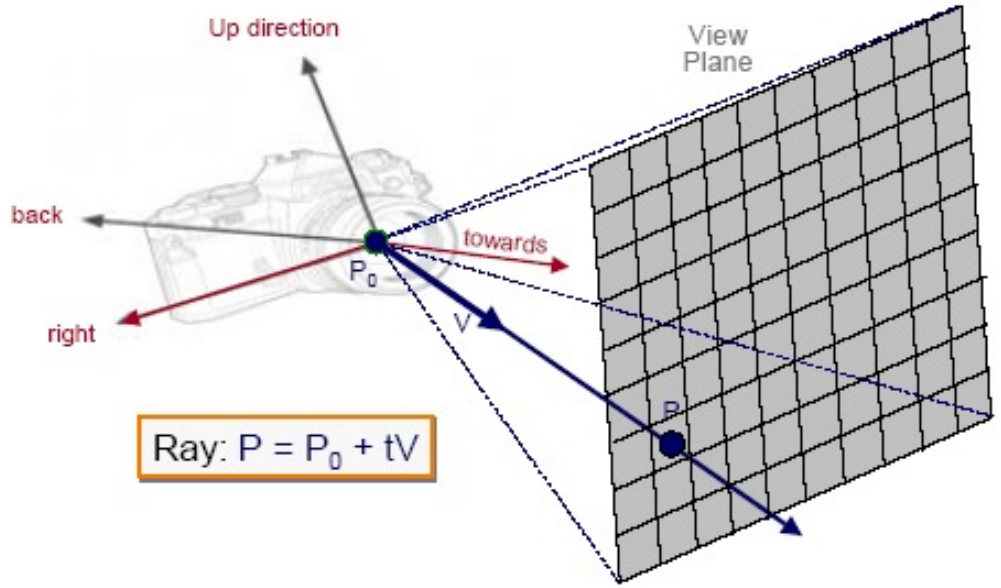
```
// ciclo principal
computeImage (viewPoint) {
   para cada ponto p in plano_imagem {
       raio = GerarRaio (viewPoint, p, PRIMARIO)
       radiance[p] =  rad (raio)
   }
}

rad (raio) {
   objecto, x = trace (raio)
   shade (x, raio, objecto)
}
```

Ray Tracing: Algoritmo

```
// intersecção mais próxima da origem do raio

trace (raio)  {
   tmin = Max_dist
   Para  todos os objectos da cena  {
       x = intersect (raio, objecto)
       dist = distancia (raio.origem, x)

       if (dist < tmin) {
              tmin = dist
              p = x
              obj = objecto                }
   }
   return (obj, p)
}
```

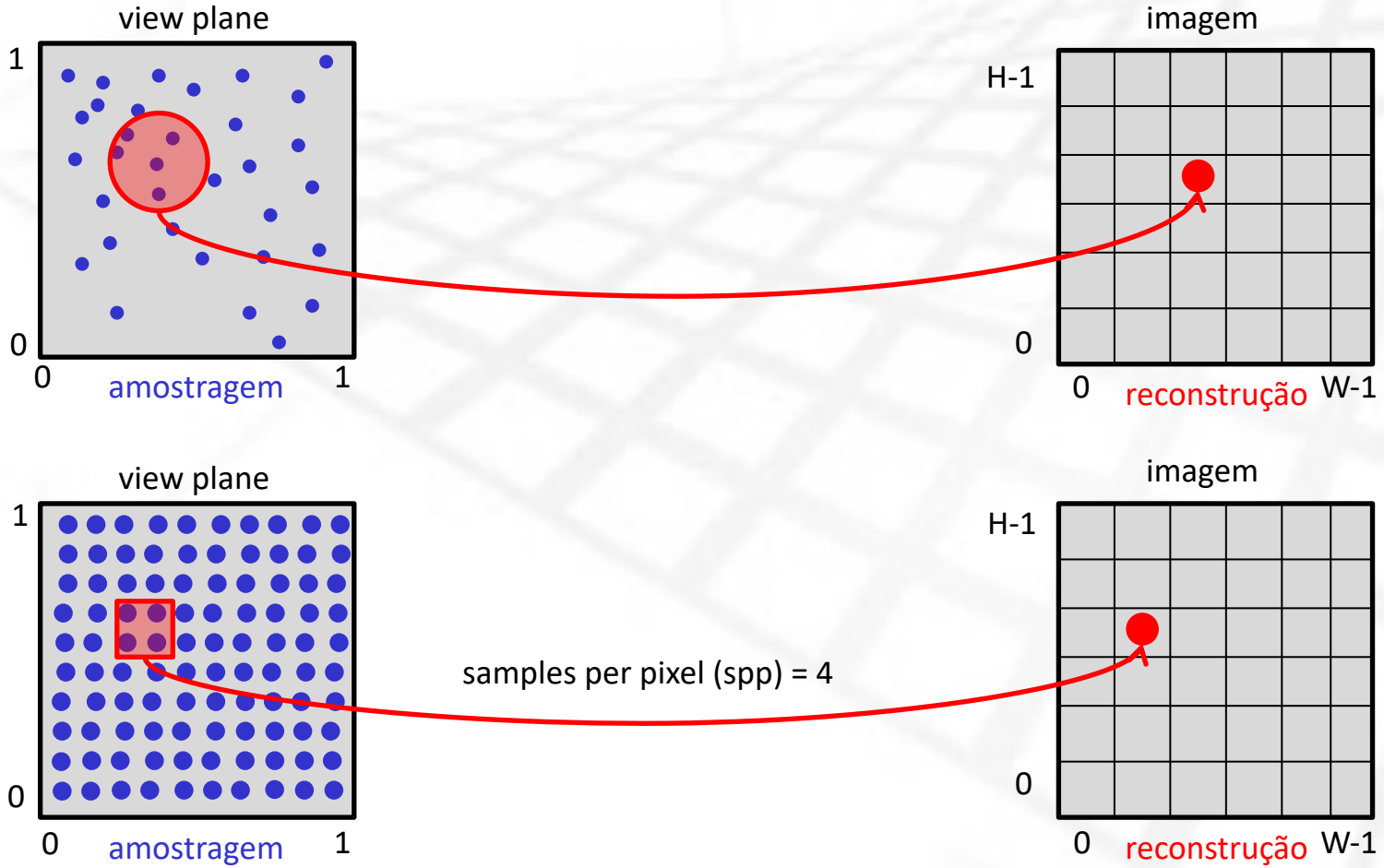
Ray Tracing: Raios Primários

- Os raios primários determinam quais os pontos visíveis directamente pelo observador. Diz-se que propagam importância, pois estes pontos passam a ser importantes para a imagem.



O número e a distribuição dos raios primários no plano da imagem determinam a frequência espacial de amostragem da visibilidade primária.

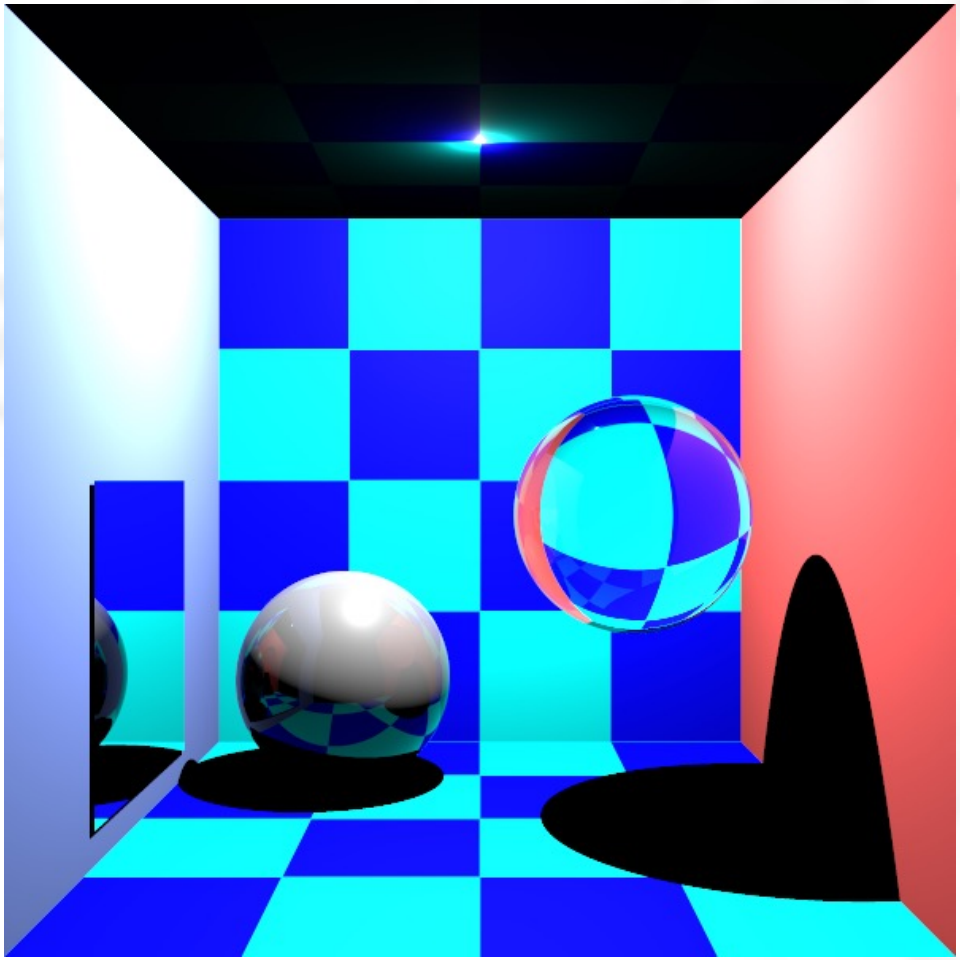
Ray Tracing: Raios Primários



Ray Tracing: *shading*

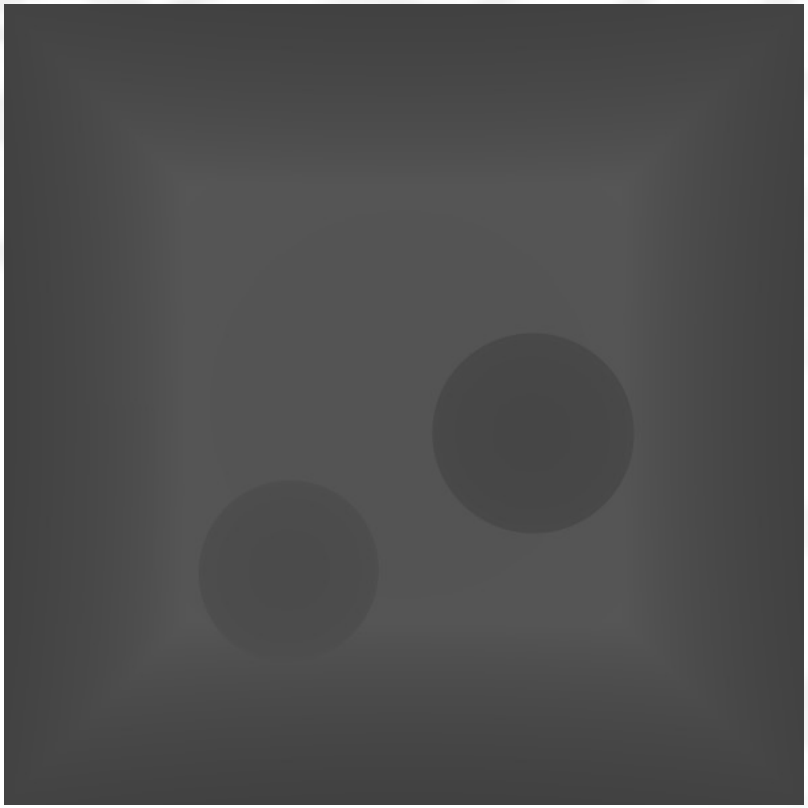
- Uma vez determinado o ponto e objecto visível é necessário calcular uma "cor" para esse ponto – operação de ***shading***:
  - Calcular uma **pseudo-cor** (ex.: visualização científica, distância, ID do objecto intersectado, etc.)
  - Calcular a contribuição das fontes de luz sem avaliar a respectiva visibilidade (**iluminação local**):
    - Inexistência de sombras
    - As placas gráficas fazem isto, usando no entanto o algoritmo de profundidade para determinar a visibilidade (*depth buffer* ou *Z-buffer*)
  - Usar funções de ***shading* recursivas** para calcular a influência de outros pontos no ponto visível (**iluminação global**)

Ray Tracing: Cornell Box



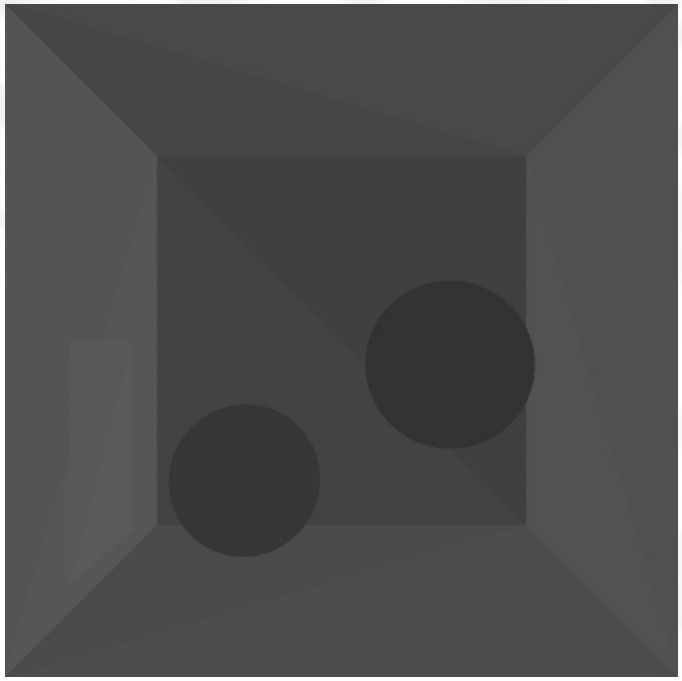
shading: pseudo-cor (depth)

```
shade (x, raio, objecto){  // depth
  return (distance(raio.o, x))
}
```



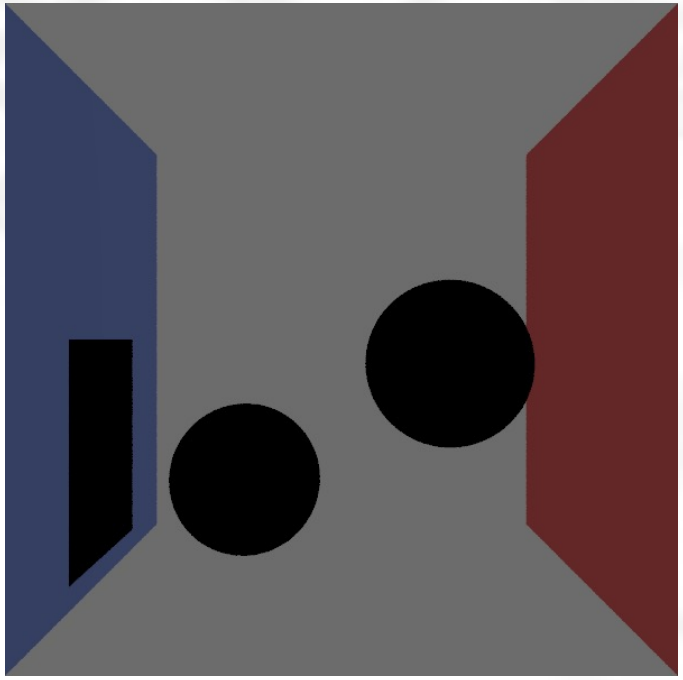
shading: pseudo-cor (ID do objecto)

```
shade (x, raio, objecto){  // object.ID
  return (objecto.ID)
}
```

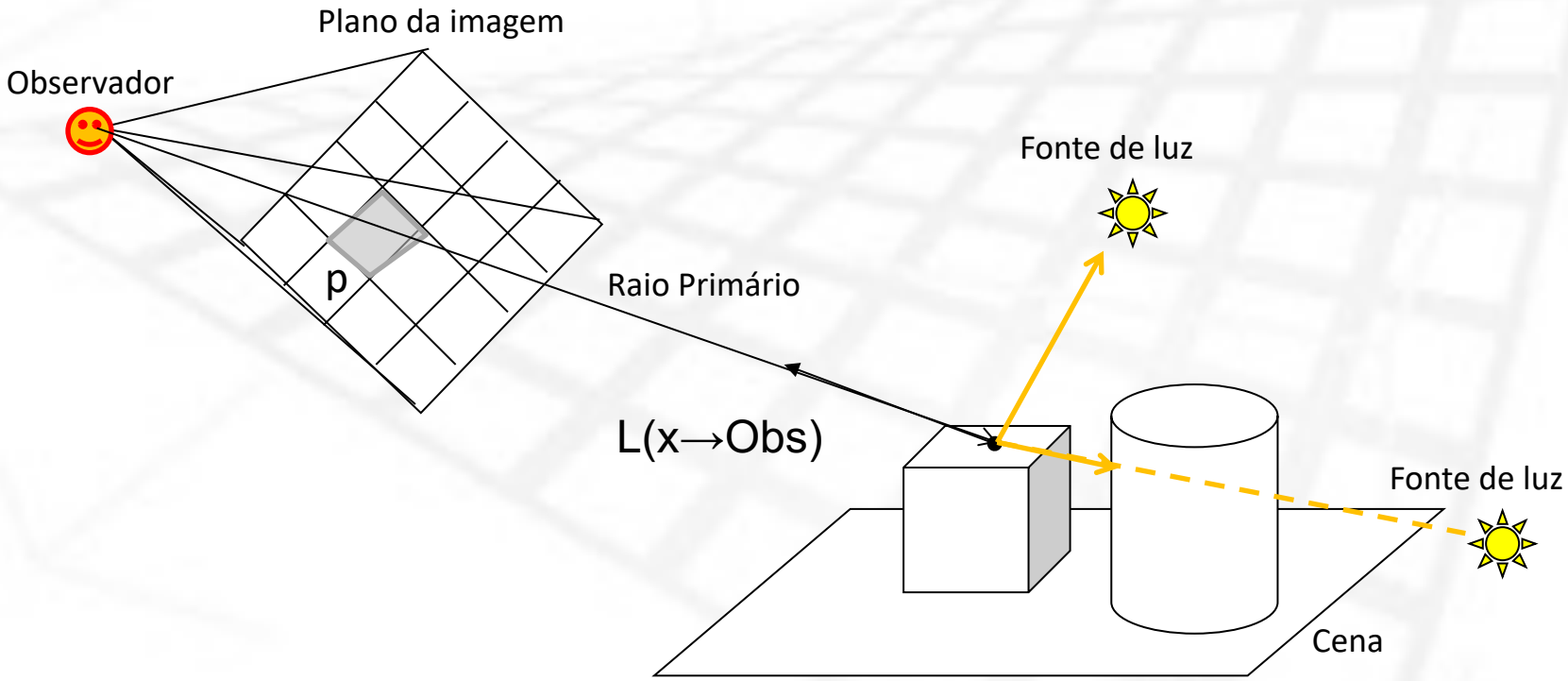


shading: pseudo-cor (BRDF)

```
shade (x, raio, objecto){  // BRDF
  return (objecto.BRDF(x, raio.dir))
}
```



Ray Tracing: Iluminação Directa



Ray Tracing: Iluminação Directa

- A maioria dos ray tracers clássicos permite aproximar as fontes de luz como fontes ideais: pontos que emitem com igual intensidade em todas as direcções.
  - Um raio disparado na direcção de x para L_i basta para determinar se a fonte pontual L_i é visível a partir do ponto *x* (chamemos a esta direcção ψ_i)

- A BRDF para este ponto e este par de direcções, $f_r(x,\Theta \leftrightarrow \psi_i)$, indica a percentagem de radiância incidente ao longo de ψ_i que é reflectida na direcção Θ.
  - Whitted sugere que o modelo de iluminação local seja o de Lambert, com um coeficiente de reflexão difusa directa para cada um dos canais: kd_R, kd_G, kd_B.
  - O modelo de Phong permite adicionar um reflexo especular à iluminação directa.

Ray Tracing: Iluminação Directa

- *V(x,y)* é a função de visibilidade:

$$V(x,y) = \begin{cases} 1 \text{ se } r(x,\Psi) = y \text{ (i.e., se y é visível a partir de x)} \\ 0 \text{ se } r(x,\Psi) != y \text{ (i.e., se y não é visível a partir de x)} \end{cases}$$

- *V(x,y)* é avaliada disparando um raio de *x* para *y*
  Se nenhum objecto for intersectado por este raio a uma distância da origem menor do que a distância da fonte de luz então *V(x,y)=1*, senão *V(x,y)=0*

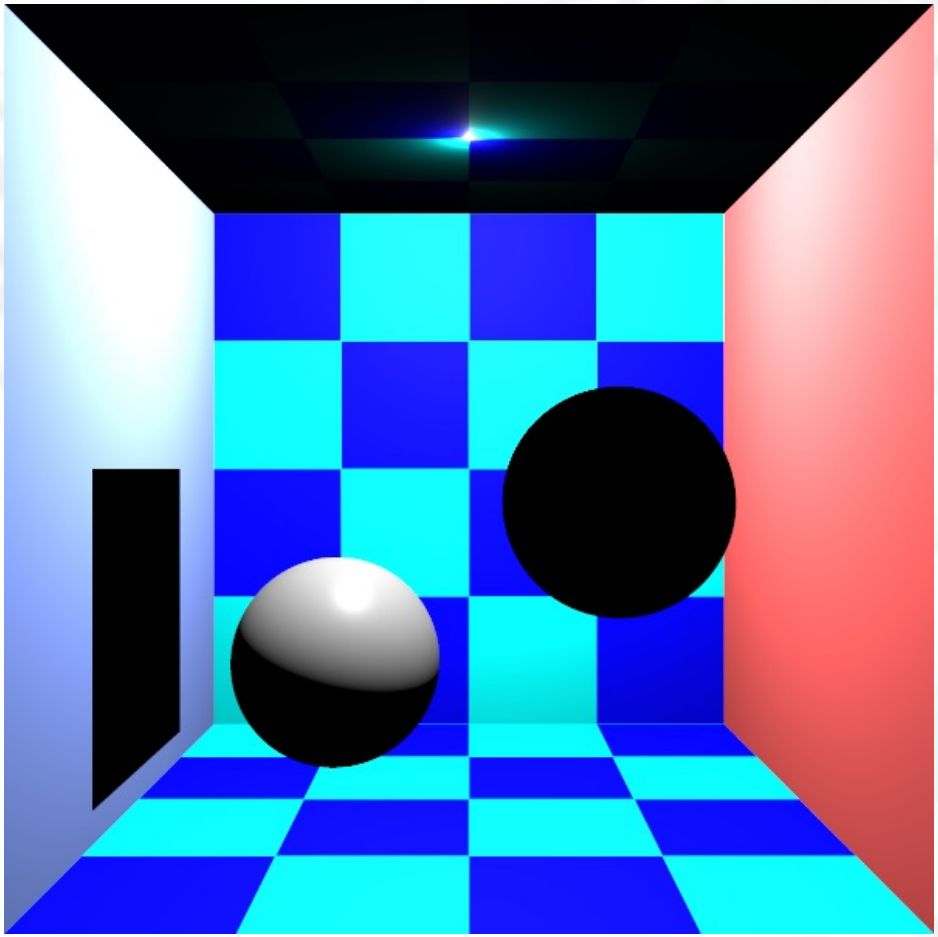
- Estes raios são designados por *shadow rays* ou *shadow feelers*

Ray Tracing: Iluminação Directa

- Se os *shadow rays* não forem disparados assume-se V(x,y)=1
- O algoritmo não calcula sombras
- A solução **NÃO** é fisicamente plausível

```
shade (x, raio, objecto) {
  radiance = directIllum_NoShadows (x, raio.dir, objecto)
  return (radiance)
}

directIllum_NoShadows (x, dir, objecto) {
   rad = 0;
   para cada fonte de luz l {
      rad += brdf (x, dir, dir_l)* L[l] * cos (Nx, dir_l)
   }
   return (rad)
}
```

Ray Tracing: Iluminação Directa



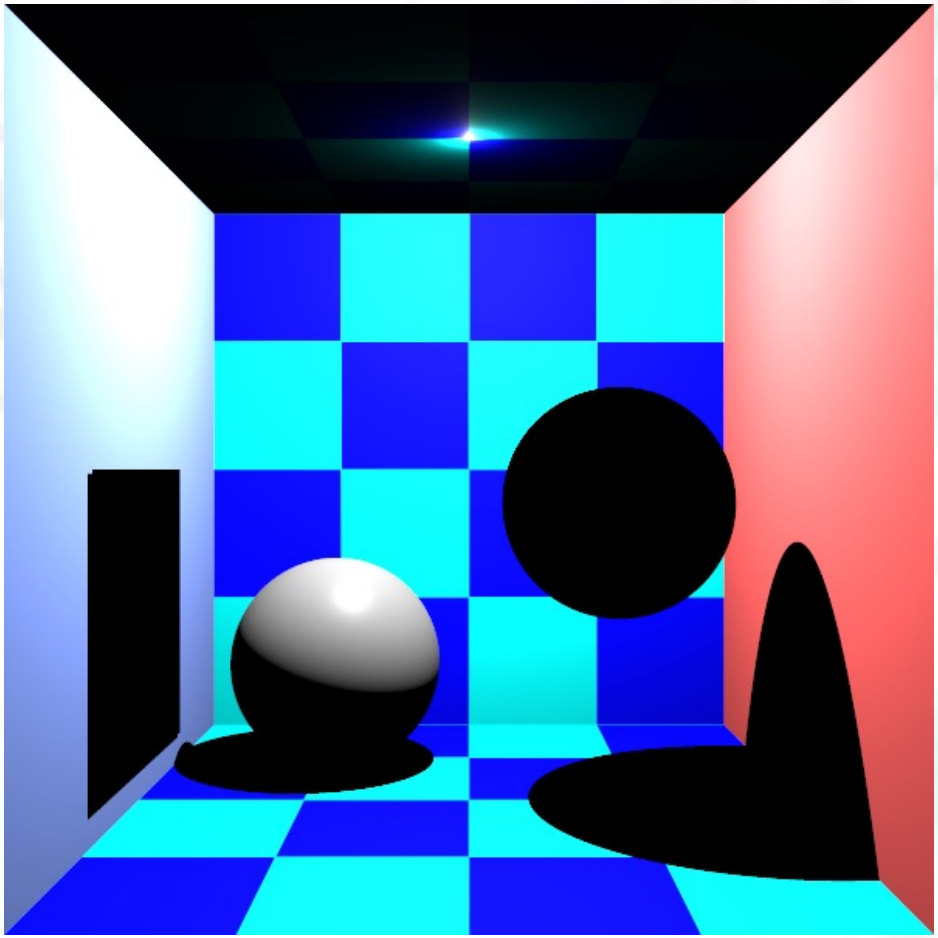
Ray Tracing: Iluminação Directa

```
shade(x, raio, objecto) {
  radiance = directIllum (x, raio.dir, objecto)
  return (radiance)
}

directIllum (x, dir, objecto) {
   rad = 0;
  para cada fonte de luz l {
       raio = GerarRaio (x, l, SHADOW)
       if (visibilidade (raio, l))
          rad += brdf (x, dir, dir_l)* L[l] * cos (Nx, dir_l)
   }
   return (rad)
}
```

Ray Tracing: Iluminação Directa

```
// visibilidade da fonte de luz

visibilidade (raio,l)  {       // V(x,y)
   tmin = distancia (raio.origem,l)
   Para  todos os objectos da cena  {
       p = intersect (raio, objecto)
       dist = distancia (raio.origem, p)
       if (dist < tmin)
             return (0)
   }
   return 1
}
```

Ray Tracing: Iluminação Directa

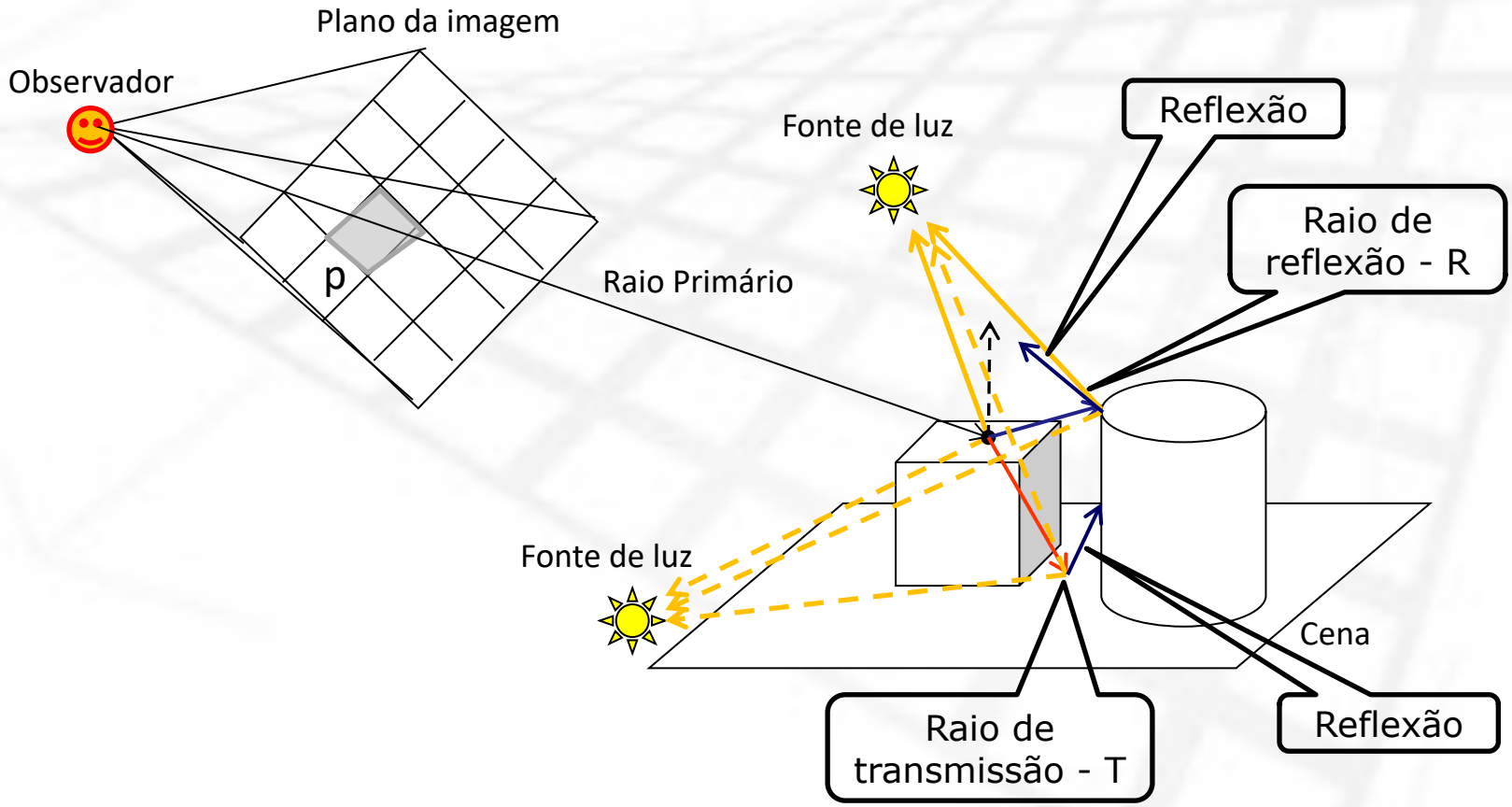


Ray Tracing: Recapitulação

- O algoritmo descrito :
  - dispara um raio primário a partir da câmara e determina o ponto *p* visível directamente
  - Com origem em *p* são disparados raios na direcção das fontes de luz

- Não há qualquer tipo de iluminação indirecta
  (luz que incide num objecto depois de ter interactuado com outro)

Ray Tracing: Iluminação Indirecta Especular



Ray Tracing: Iluminação Indirecta Especular

- Reflexão especular

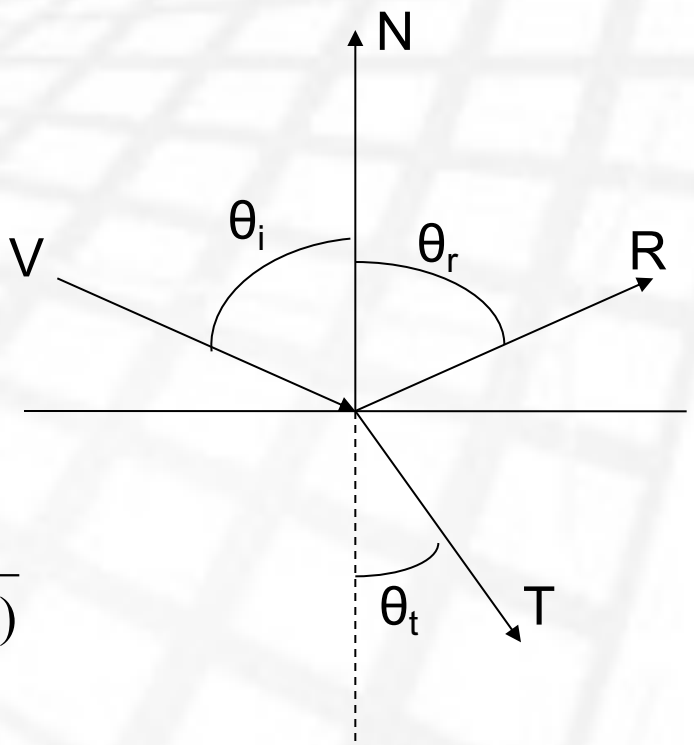
$$R = 2N(N \cdot V) - V$$

- Transmissão especular
  (lei de Snell)

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

$$\cos\theta_t = \sqrt{1 - \sin^2\theta_t} = \sqrt{1 - \eta^{-2}(1 - (V \cdot N)^2)}$$

$$T = \frac{V}{\eta} - (\cos\theta_t - \frac{(V \cdot N)}{\eta})N$$



Ray Tracing: Iluminação Indirecta Especular

É aproximada apenas pelas direcções R e T.

A BRDF é aproximada pelas constantes:

- $k_{sg}(\lambda)$ – coeficiente de reflexão especular global
- $k_{tg}(\lambda)$ – coeficiente de transmissão especular global

$$L_{indirecta,\lambda}(x \rightarrow \Theta) = k_{sg}(\lambda) L_{r,\lambda}(r(x,R) \rightarrow -R) \cos(N_x, R)$$
$$+ k_{tg}(\lambda) L_{r,\lambda}(r(x,T) \rightarrow -T) \cos(N_x, T)$$

Ray Tracing: Iluminação Indirecta Especular

- Para calcular a radiância incidente em *x* ao longo de cada uma das direcções R e T devem ser enviados raios secundários ao longo de cada uma destas direcções.

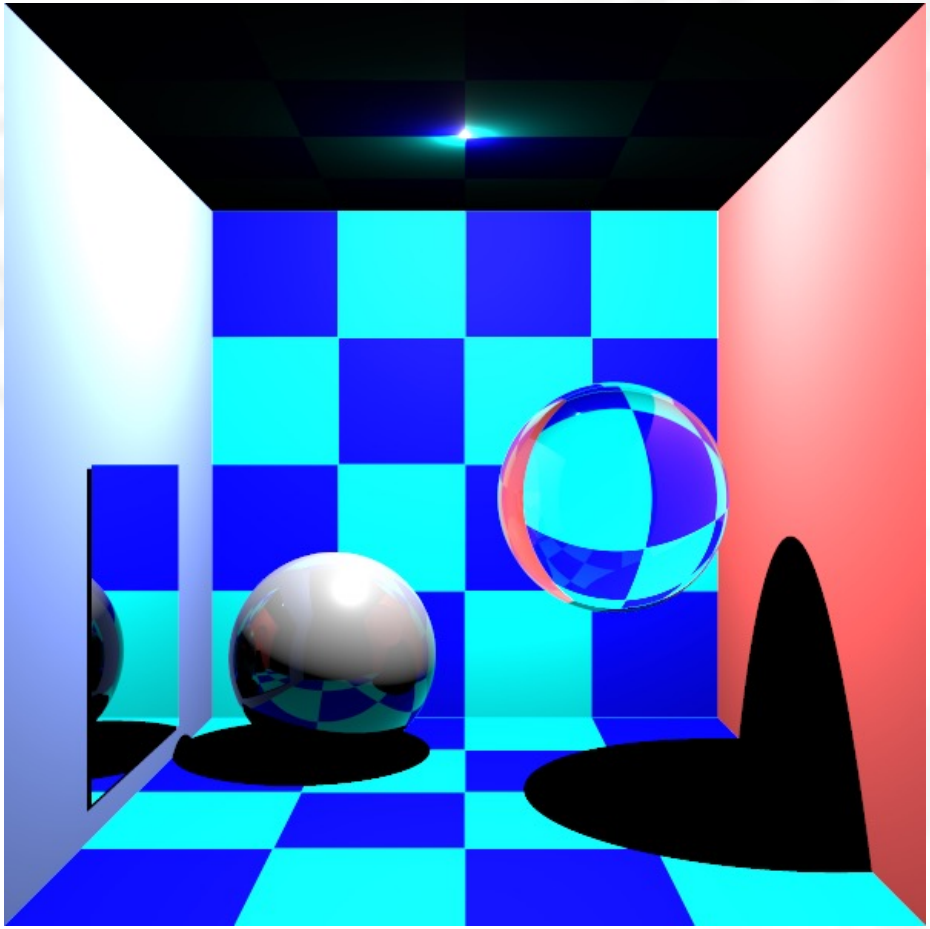
- O processamento dos raios secundários é em tudo equivalente aos raios primários, fazendo do *ray tracing* um algoritmo recursivo que gera uma árvore de raios.

- É necessário um critério de paragem para que a árvore não tenha profundidade infinita:
  - Terminar ao atingir uma determinada profundidade
  - Terminar quando a contribuição de um raio for inferior a um determinado limite
  - Decidir de forma estocástica (Roleta Russa)

Ray Tracing: Iluminação Indirecta Especular

```
shade (x, raio, objecto, depth) {
  rad = directIllum (x, raio.dir, objecto)

   if (depth < MAX_DEPTH) {
     if (ksg > 0)  {  // reflexão especular
        raioR = GerarRaio (x, Rg, REFLEXAO)
        objR, p = trace (raioR)
        rad += ksg*cos(N,raioR)* shade (p,raioR,objR, depth++)}

     if (ktg > 0)  {  // transmissão especular
        raioT = GerarRaio (x, Tg, TRANSMISSAO)
        objT, p = trace (raioT)
        rad += ktg*cos(N,raioT)* shade (p,raioT,objT, depth++)}
   }
  return (rad)
 }
```

Ray Tracing: Iluminação Indirecta Especular

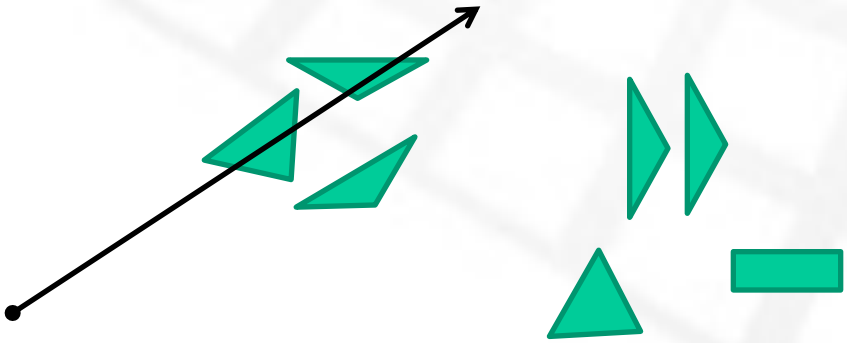


Ray Tracing: Complexidade

- O algoritmo usado para determinar a visibilidade ao longo de um raio requer que cada raio seja intersectado com TODAS as primitivas geométricas da cena.
- O tempo para cada raio é, portanto, linear com o número de primitivas N:

$$T_{\text{ray}} = \mathrm{O}(N)$$

- No entanto, cada raio não passa na proximidade da maioria das primitivas



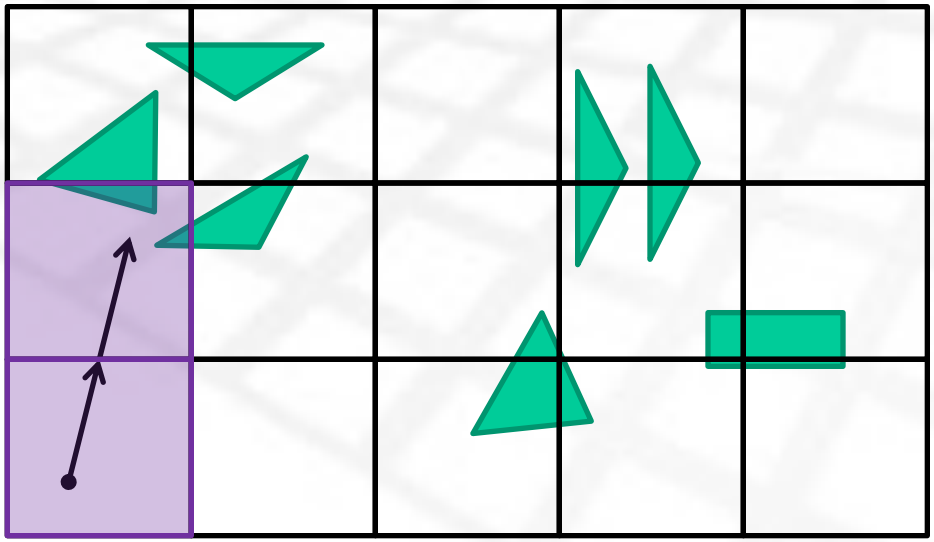
Ray Tracing: Estruturas de Aceleração

- O objectivo das estruturas de aceleração é diminuir o número de intersecções por raio.
- Isto é conseguido:
  1. Permitindo a rejeição rápida e simultânea de grupos de primitivas
  2. Se possível, ordenando o processo de procura (intersecções), tal que as primitivas mais próximas da origem do raio sejam processadas primeiro, evitando processar as mais distantes se for encontrada uma intersecção

- Abordagens:
  - SUBDIVISÃO DO ESPAÇO: grelhas regulares, octrees, kd-tree
    Permitem aplicar os critérios 1 e 2
  - SUBDIVISÃO DOS OBJECTOS: bounding volume hierarchy (BVH)
    Permitem aplicar apenas o critério 1

Ray Tracing: Grelha Regular

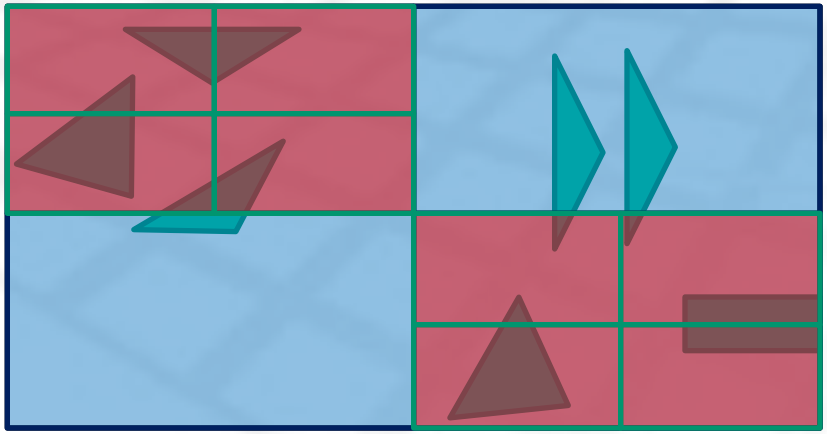
- O espaço 3D é particionado impondo uma grelha regular que o subdivide em voxels (*volume elements*). Todos os voxels têm a mesma dimensão.



- Construção muito rápida
- Travessia pouco eficiente devido à má distribuição das primitivas pelos *voxels*

Ray Tracing: Octree

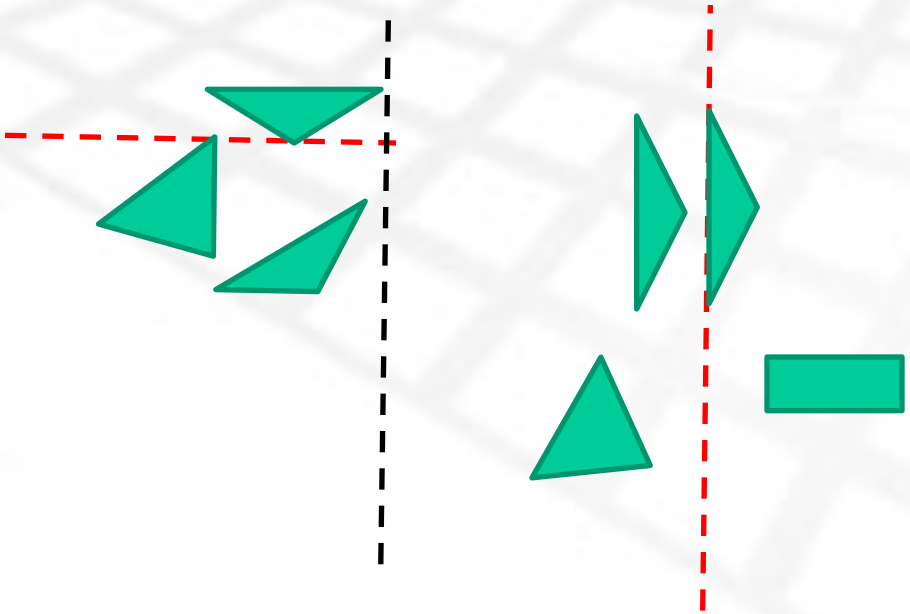
- O espaço é hierarquicamente e adaptativamente subdividido em 8 voxels



- Compromisso entre tempo de construção e eficiência da travessia

Ray Tracing: Kd-tree

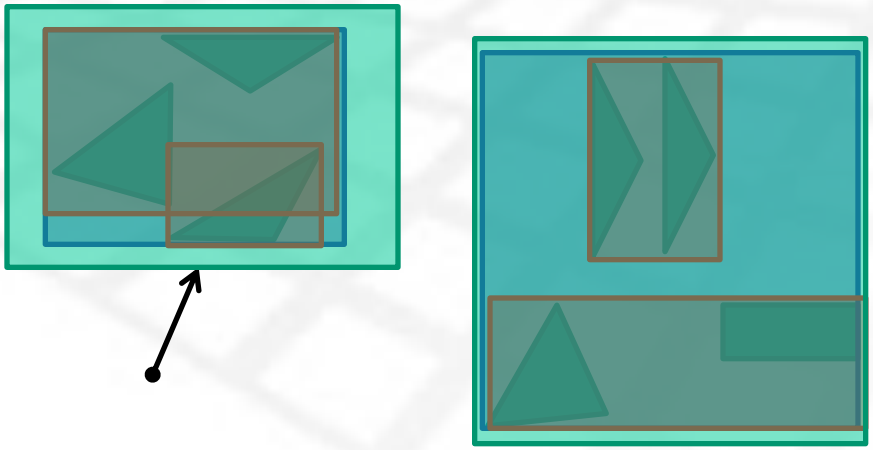
- O espaço é subdividido em 2 por um plano. Cada um dos sub-espaços resultantes é depois subdividido da mesma forma, até atingir um determinado critério de paragem



- Travessia mais eficiente se o critério de subdivisão for apropriado (e.g., SAH)
- Quanto mais sofisticado for o critério de subdivisão maior o tempo necessário para a construir

Ray Tracing: Bounding Volume Hierarchy

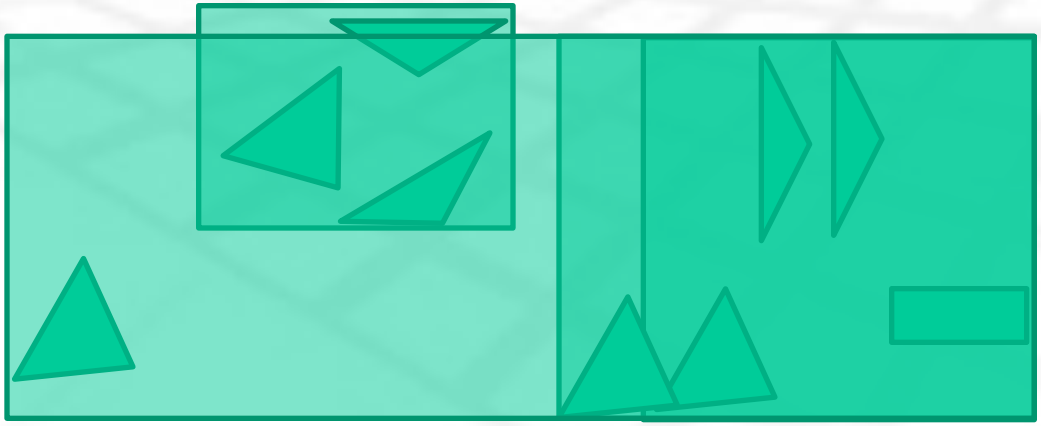
- Os objectos são agrupados dentro de Bounding Volumes. Cada um destes grupos é depois hierarquicamente subdividido por outros volumes



- Não ordena o espaço
- Construção semelhante à kd-tree
- Travessia ligeiramente inferior à kd-tree

Ray Tracing: Geometria Dinâmica

- Qual a estrutura mais indicada se a geometria se move?



- BVH permite reaproveitamento da hierarquia (topologia da árvore), com ajustamento das dimensões dos volumes
- Com grandes deformações da geometria a hierarquia inicial deixa de ser apropriada para a distribuição das primitivas, exigindo uma reconstrução completa da BVH

Ray Tracing: Complexidade

- A complexidade do *ray tracing* com uma estrura de aceleração apropriada é logarítmica com o número de primitivas geométricas N:

$$T_{\text{ray}} = \text{O}(\log N)$$

- O tempo de construção depende do critério de subdivisão do espaço / agrupamento das primitivas

- Critérios sofisticados (e.g., SAH) resultam em travessias eficientes, mas exigem tempos de construção muito elevados

- O tempo de reconstrução/reajustamento de uma estrutura de aceleração pode impedir a sua utilização em contextos interactivos.

Trajectos de luz

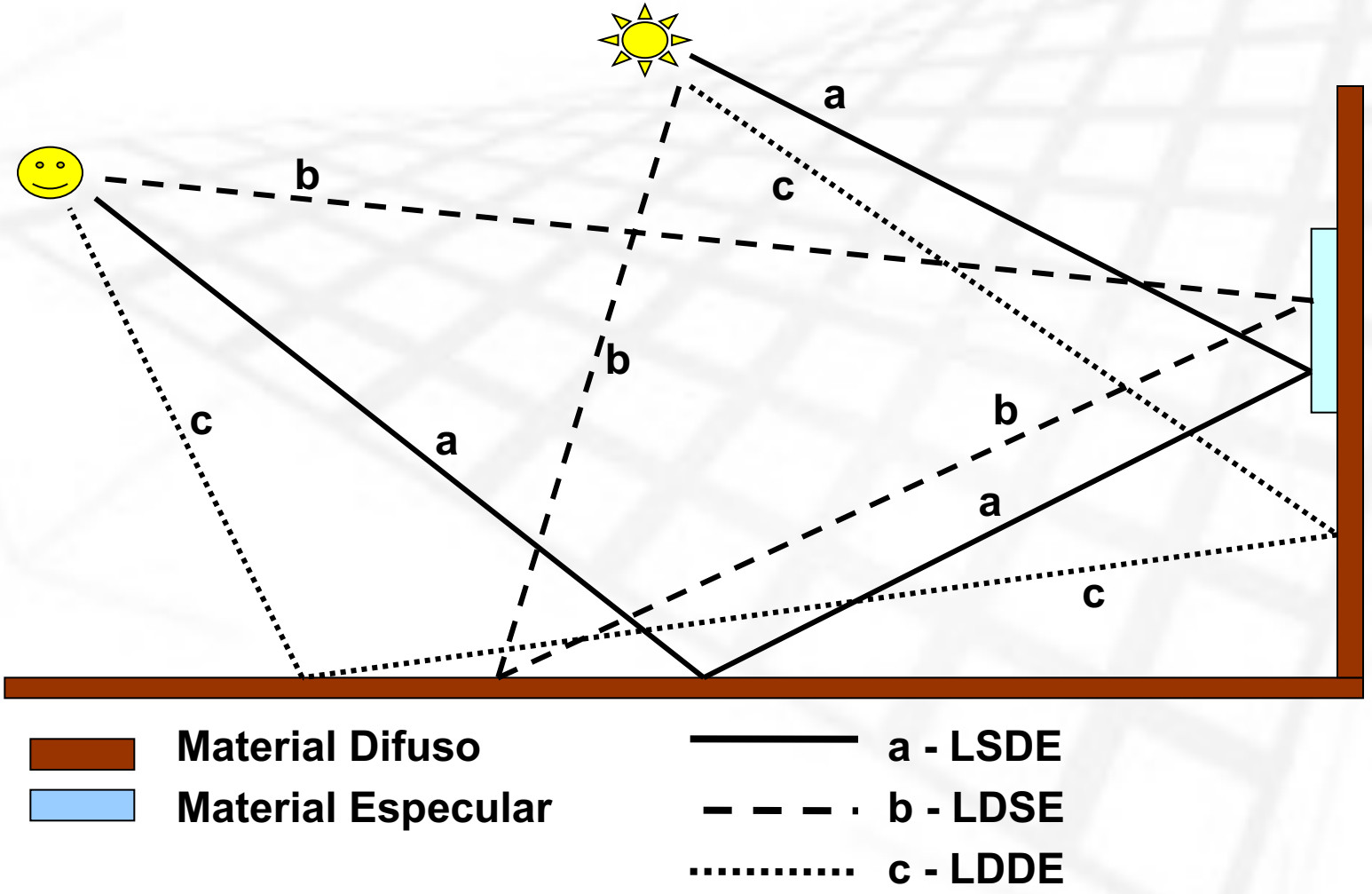
- Sendo o transporte de luz aproximado pela óptica geométrica, podemos conceber que cada fotão percorre um trajecto desde a fonte de luz até ao seu destino final

- Este trajecto é composto por segmentos de recta direccionais, representando os extremos interacções do fotão com um objecto.

- A interacção pode ser difusa (D) ou especular (S)

- O conjunto de interacções é representado por uma string com origem na fonte de luz (L) e a terminar no ponto onde o fotão é absorvido ou no observador (E)

Trajectos de luz



Trajectos de luz

- A árvore de raios gerada pelo *ray tracing* de Whitted começa no observador e termina sempre que encontrar um material difuso.

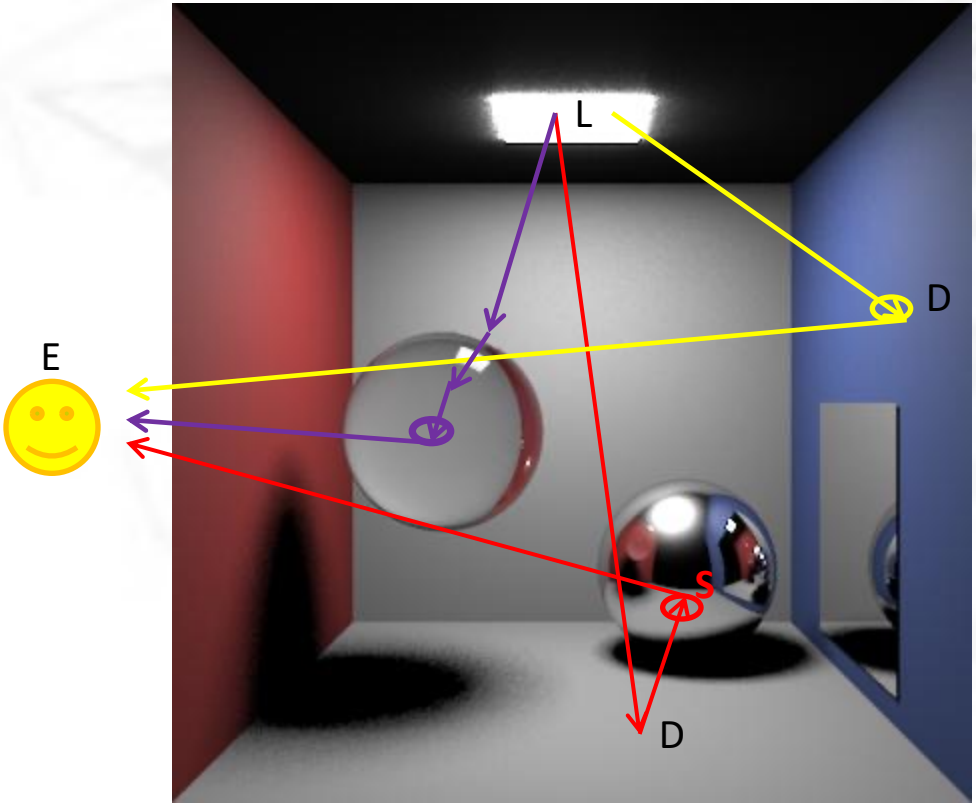
- Os trajectos simulados são portanto sempre do tipo

  L[D][S*]E

- Os caminhos ‘a’ e ‘c’ do acetato anterior não são simulados

- O caminho ‘a’ resultaria na projecção pelo espelho do reflexo da fonte de luz no chão difuso. O *ray tracing* clássico não modela este fenómeno.

Trajectos de luz: *Whitted ray tracing*

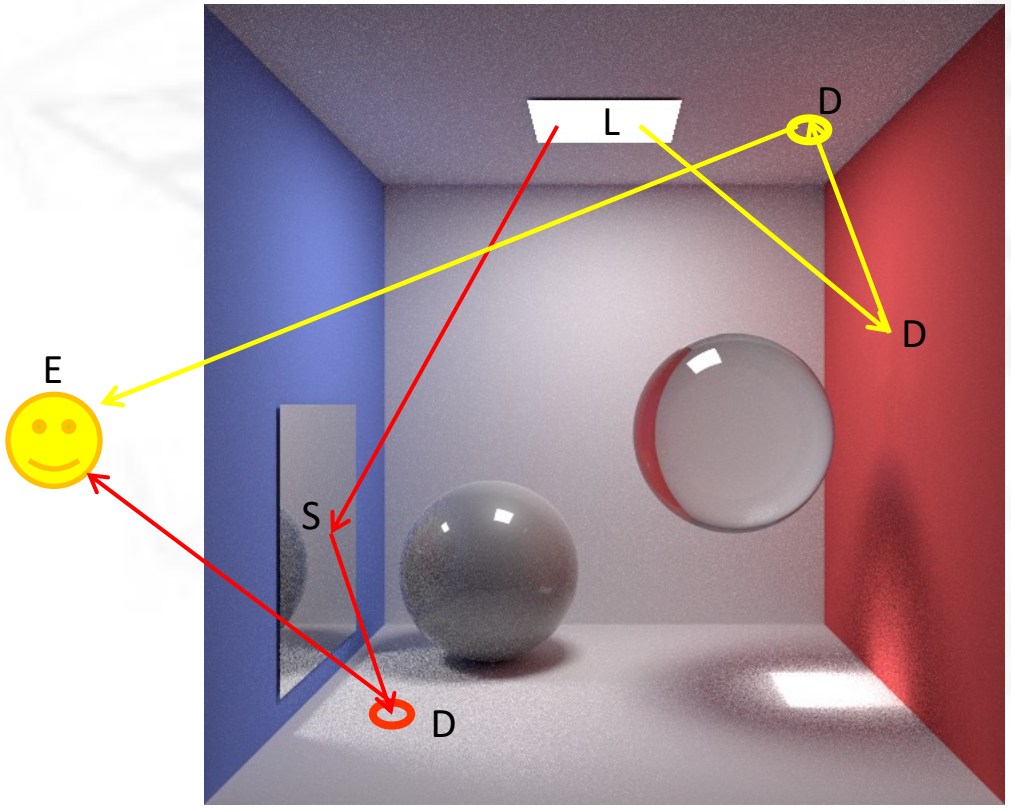


LDSE

LDE

LDSSE

Trajectos de luz: *path tracing*



LSDE

LDDE

Os trajectos de luz – seguidos a partir do observador – não terminam em materiais difusos. Podem continuar pois é sempre seleccionada uma direcção para amostrar.