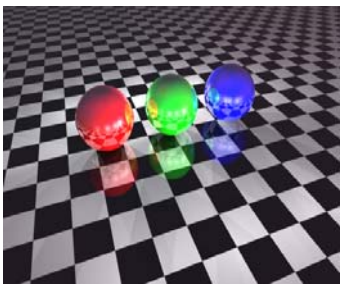
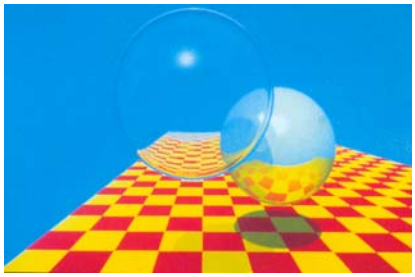
# Modelos de iluminação global

Exemplos de imagens produzidas com ***ray-tracing***

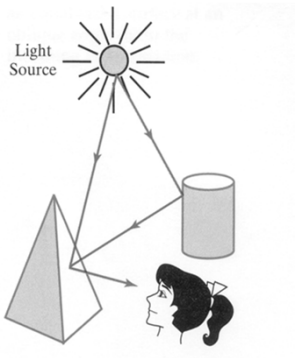






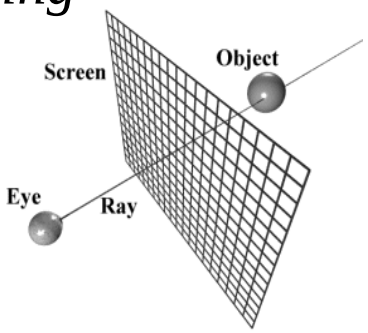
# Modelos de Iluminação Global

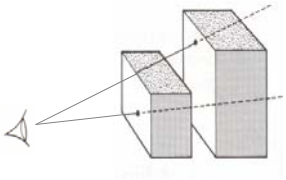
Os **modelos de iluminação global** tratam de forma explícita a interacção luminosa entre objectos.



# *Ray-casting*

- Se se considerar a linha de vista a partir de um *pixel* no plano de visualização até à cena, é possível determinar que objectos são intersectados
- Este método chama-se ***ray casting*** e **inverte o sentido do trajecto dos raios de luz**, passando agora o observador a ser a fonte
- Baseia-se nos métodos de óptica geométrica que determinam os **percursos dos raios de luz**



Quando se usa **projecção perspectiva**, os raios divergem do centro de projecção, passam pelo **centro de um *pixel*** e continuam através da cena



# *Ray-tracing*

Esforço computacional elevado devido ao cálculo de grande nº de intersecções

Ex:

- imagem 1024 x 1024 *pixels*
- 100 superfícies
- Pode ser necessário calcular 100 milhões de intersecções.

# *Ray-tracing*

Exemplo de estratégias de optimização:

- Reduzir o esforço computacional através da utilização de volumes envolventes (ex: *bounding boxes*)

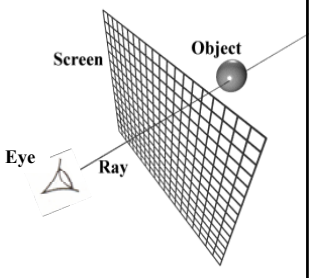
  Se um raio não intersecta o volume envolvente também não intersecta o(s) modelo(s) envolvido(s)



# *Ray-casting*

O ***ray-casting*** pode ser usado para a eliminação de invisíveis (algoritmo do espaço imagem)

```
select center of projection and window on viewplane;
for ( each scan line in image ) {
    for ( each pixel in scan line ) {
        determine ray from center of projection through pixel;
        for ( each object in scene ) {
            if (object is intersected and is closest considered thus far)
                record intersection and object name;
        }
        set pixel's color to that at closest object intersection;
    }
}
```

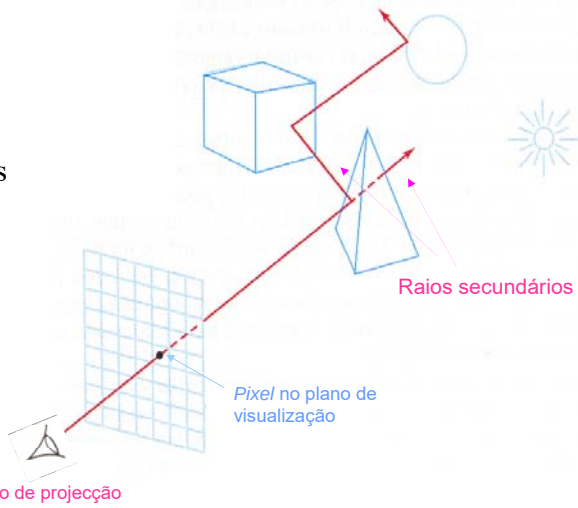



**" a ray-casting function: shoots a ray into a scene and finds the first surface it encounters"** (*slide seguinte)

# *Ray-tracing*

Podem também lançar-se raios secundários reflectidos ou transmitidos a partir do ponto de intersecção mais à frente na cena (*slide anterior)

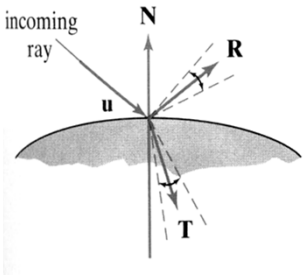
No caso de haver raios secundários o algoritmo chama-se *ray-tracing* (lançamento recursivo de raios)

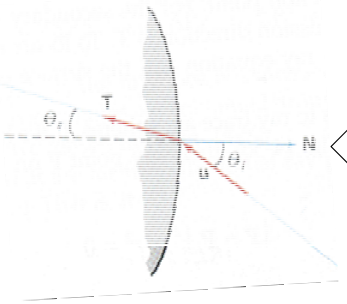





# *Ray-tracing*

Raios secundários:
**reflectidos (R)** e **transmitidos (T)**




**Raio transmitido ou refractado (T)** – o raio sofre um desvio ao entrar num meio em que a velocidade de propagação da luz é diferente

# *Ray-tracing*

- O algoritmo ***ray-tracing*** combina:
  - Eliminação de superfícies ocultas
  - *Shading* devido a iluminação directa
  - *Shading* devido a iluminação global
  - Determinação de sombras

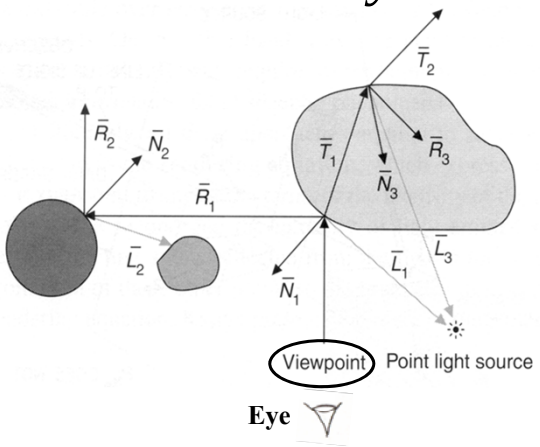
O algoritmo ***ray-tracing*** **trata explicitamente as interacções luminosas entre os objectos** → **modelo de iluminação global**



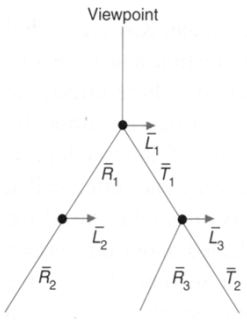
# *Ray-tracing*

- Descrição resumida do algoritmo *ray-tracing*:
  - Traçar um raio a partir da posição do observador
  - Intersectá-lo com os modelos da cena
  - Determinar o ponto de intersecção mais próximo
  - Calcular a iluminação nesse ponto de intersecção
  - Atribuir a cor ao pixel.

# *Ray-tracing*

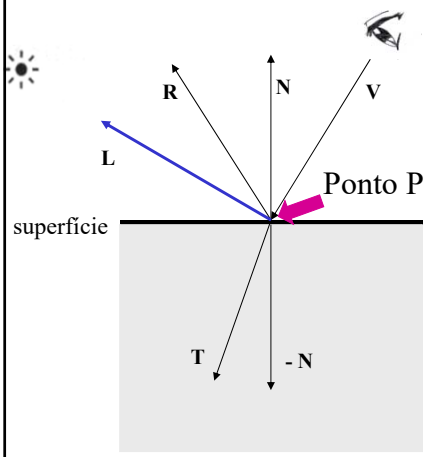



**Árvore de raios** ⇨






# *Ray-tracing*



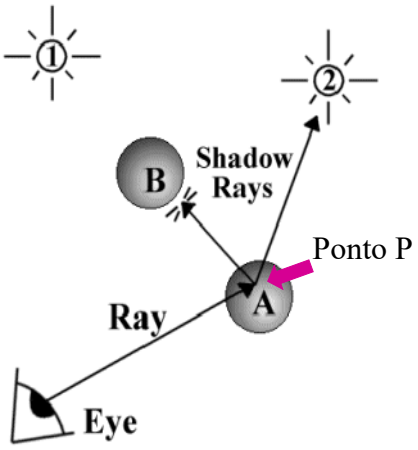

Para calcular **sombras** é lançado um raio adicional a partir do ponto de intersecção com a superfície (ponto P) para cada uma das fontes de luz presentes na cena (raio L- *shadow ray*).

Se o raio encontrar um objecto opaco no seu percurso até à fonte de luz, então o ponto P está na sombra em relação a esta fonte de luz.

- V designa-se por **raio principal**
- R, L e T designam-se por **raios secundários**

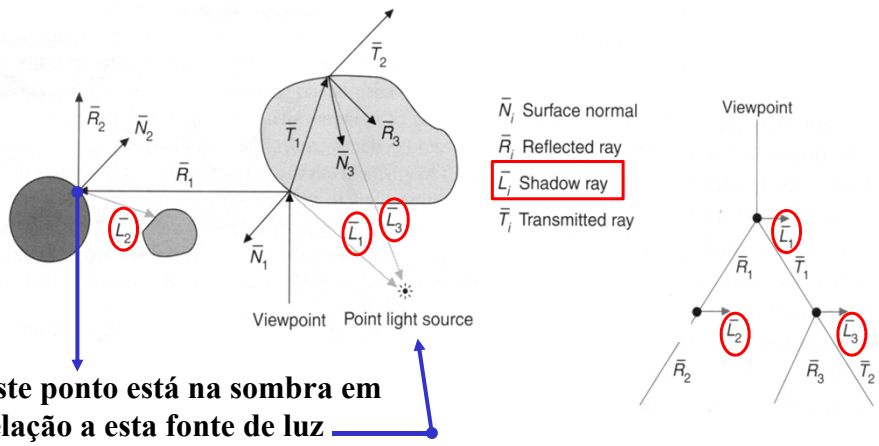
# *Ray-tracing*

**Exemplo:** O ponto P está na sombra em relação à fonte de Luz 1






# *Ray-tracing*

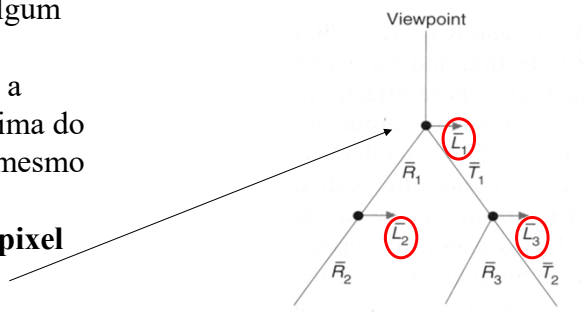



**Este ponto está na sombra em relação a esta fonte de luz**

# *Ray-tracing*

Se um determinado raio não intersectar nenhum objecto no seu trajecto, é atribuída a **<u>cor do fundo da cena</u>** ao pixel por onde o raio passa.

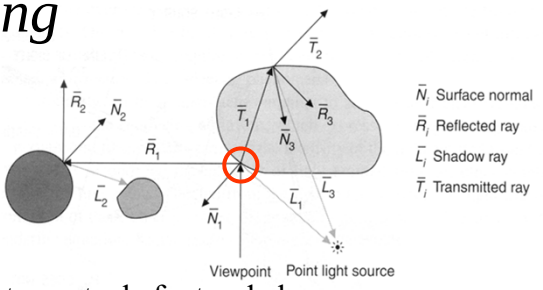
Se o raio intersectar algum objecto, e após ter-se confirmado que esta é a intersecção mais próxima do observador para esse mesmo raio, é necessário **determinar a cor do pixel correspondente**…






# *Ray-tracing*

a **iluminação no ponto da cena que o raio atinge:**




- pode ser proveniente directamente de fontes de luz,
- pode ser luz proveniente de outro objecto que por reflexão ilumina o ponto que estamos a analisar,
- pode ser luz refractada transmitida através do objecto e que assim ilumina o ponto, ou
- pode ainda ser uma <u>combinação de mais do que uma destas formas de iluminação</u> (a situação mais comum).

# *Ray-tracing*

A **cor de um pixel** corresponde à soma de todas as contribuições que são calculadas em todos os nós da árvore gerada pelo lançamento de raios secundários

O algoritmo ray tracing foi proposto por Whitted em 1980



# Equação de Iluminação de Whitted

Com **m** fontes de luz:

$$\mathbf{I = I_a k_a + \sum_{1 \le i \le m} ( S_i I_{pi} f_{ati} [ k_d (L_i . N) + k_s (R_i . V)^n ] ) + k_s I_r + k_t I_t}$$

Sombra: $\mathbf{S_i}$

- 0 Se a luz proveniente da fonte de luz pontual i não alcança o ponto (por existir um objecto opaco entre a fonte de luz e o ponto)
- 1 Caso contrário

# Equação de Iluminação de Whitted

$$I = I_a k_a + \sum_{1 \le i \le m} ( S_i I_{pi} f_{ati} [ k_d (L_i.N) + k_s (R_i.V)^n ] ) + k_s I_r + k_t I_t$$

$k_t$- coeficiente de transmissão (valor entre 0 e 1)

Sombra: $S_i$ Whitted propôs usar uma função contínua, que depende do $k_t$ de cada objecto no trajecto do raio sombra (até à fonte de luz); um objecto opaco impede totalmente a passagem da luz, mas um objecto transparente só impede a sua passagem parcialmente.



# Equação de Iluminação de Whitted

$$I = I_a k_a + \sum_{1 \le i \le m} ( S_i I_{pi} f_{ati} [ k_d (L_i.N) + k_s (R_i.V)^n ] ) + k_s I_r + k_t I_t$$

$I_r$- intensidade do raio reflectido

$I_t$- intensidade do raio transmitido (se o objecto for transparente)

$k_t$- coeficiente de transmissão (valor entre 0 e 1)

**$I_r$ e $I_t$ são determinados recursivamente aplicando esta equação na superfície mais próxima que os raios refletido e transmitido intersectam**.

# *Ray-tracing*

**Algorithm: classical recursive ray tracing**

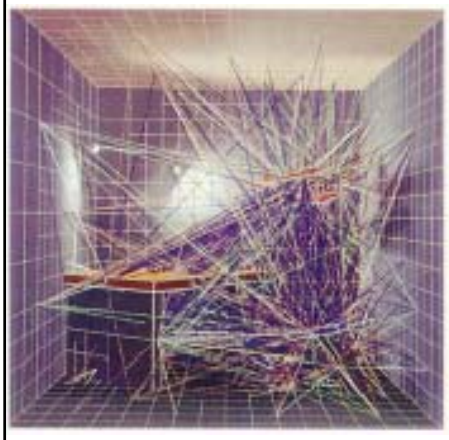
```
For each pixel in image {
  Create ray from eyepoint passing through this pixel
  Initialize NearestT to INFINITY and NearestObject to NULL

  For every object in scene {
     If ray intersects this object {
        If t of intersection is less than NearestT {
           Set NearestT to t of the intersection
           Set NearestObject to this object
        }
     }
  }

  If NearestObject is NULL {
     Fill this pixel with background color
  } Else {
     Shoot a ray to each light source to check if in shadow
     If surface is reflective, generate reflection ray: recurse
     If surface is transparent, generate refraction ray: recurse
     Use NearestObject and NearestT to compute shading function
     Fill this pixel with color result of shading function
  }
}
```



# *Ray-tracing*



Número de raios luminosos muito elevado…

Definem-se condições de terminação do algoritmo

# *Ray-tracing*

O processo de lançamento de raios secundários (reflectidos ou transmitidos) termina quando:

- O raio não intersecta nenhuma superfície
- O raio intersecta uma fonte de luz que não é uma superfície reflectora
- O nível de profundidade da árvore construída com os sucessivos raios atingiu o limite máximo pré-definido
- (...)



# Radiosidade

Baseia-se nos princípios de radiação térmica entre superfícies para a **transferência de calor entre estas.**

**Adequado para cenas em que predominam superfícies reflectoras difusas:**

- **a energia luminosa incidente numa superfície é reflectida com igual intensidade em todas as direções;**
- **O seu aspecto não depende da posição do observador**

Calcula a **taxa de energia luminosa** total que é libertada pelas superfícies

# Radiosidade

A **radiosidade** trata de uma cena dividindo-a geometricamente num conjunto de bocados (*patches*). Em cada um utilizam-se equações lineares de modo a calcular-se como a luz viaja entre cada par de *patches* (método pesado em termos de processamento).
Cada *patch* é pintado de uma só cor.



# Radiosidade

Para um *patch* (superfície) j:

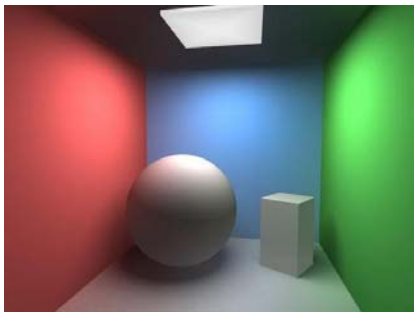
$$B_j = Ej + \rho_j H_j$$

A quantidade de energia reflectida é o produto entre a quantidade de energia incidente na superfície e a constante de reflexão

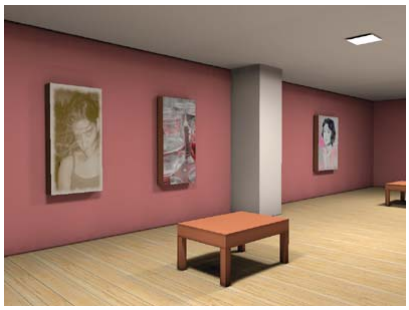
$B_j$ – radiosidade
Ej – **energia emitida** pela superfície j
$\rho_j$ – constante de reflexão da superfície j
$H_j$ – energia incidente na superfície j, vinda de outros *patches* (é uma soma e depende da posição relativa entre os *patches*)

# Radiosidade

Produz imagens com sombras suaves e graduais



http://www.polycount.com/forum/showthread.php?t=135481



http://http.developer.nvidia.com/GPUGems2/gpugems2_chapter39.html