

Tema 6 - Realismo Visual

**Parte 2 - TECNICAS
globais e locais**

UFF - 2020

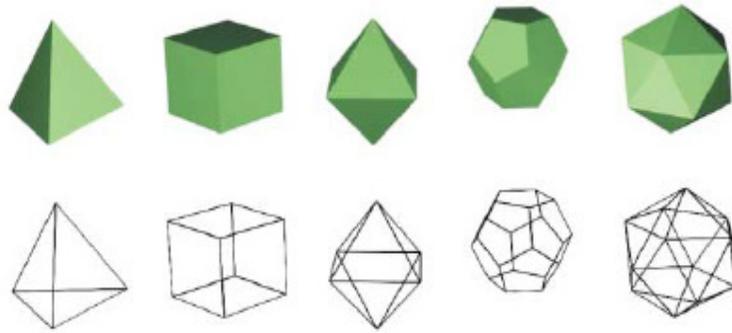


Algoritmos em coordenadas do mundo

Como melhorar o realismo de Objeto na cena?



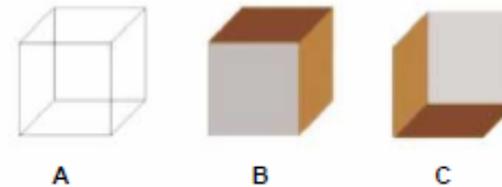
imagine que temos objeto e um observador e queremos tornar realista, esse objeto para ser visto pelo observador...



Considerando a Geometria do objeto para o observador da cena

Iremos da Representação 3D (*wire frame*)

Para algo mais realista com a Eliminação de partes não visíveis pelo observador deste objeto.



Representação aramada em A e suas possíveis interpretações em B e C.

Eliminação de partes não visíveis

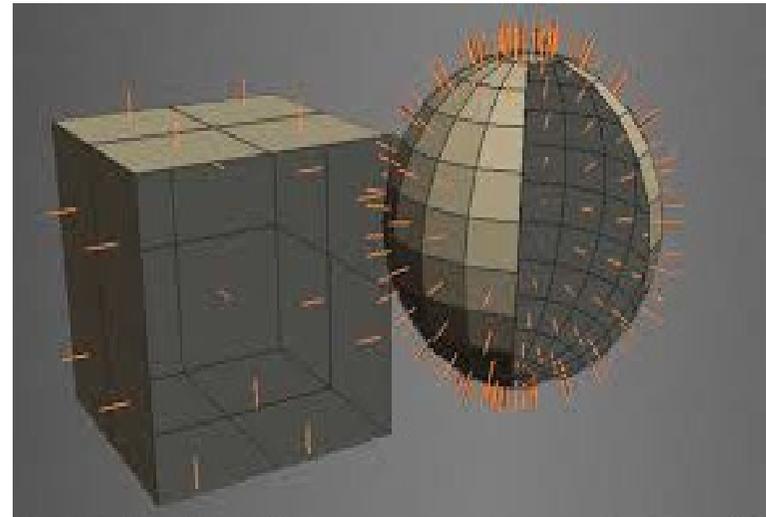
Técnicas de visibilidade

Back face culling (método de Roberts ou teste da normal)

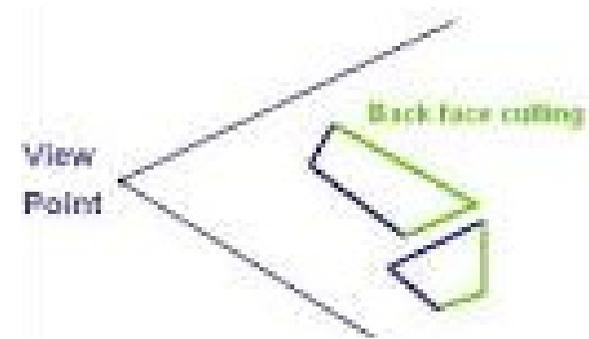
Priority fill ou *painter's algorithm*

Z- buffer
(*mim Max*)

Ray casting
(*Ray tracing simplificado*
ou *aproximado*)



Para obj. convexos:

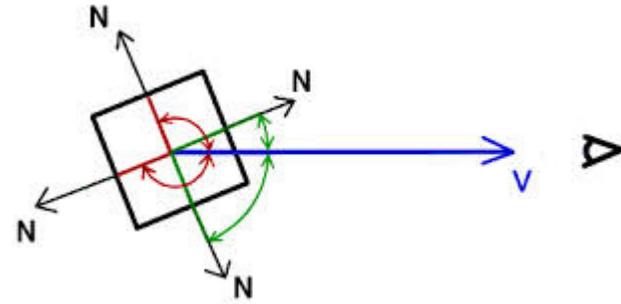


Back face culling

Em CG back-face culling determina quando a face de um objeto será visível.

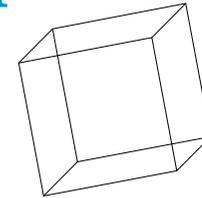
Esse processo torna o rendering mais eficiente pois reduz o número de polígonos a ser desenhado (só os visíveis continuam a ser tratados)

Back face culling



Idéia básica:

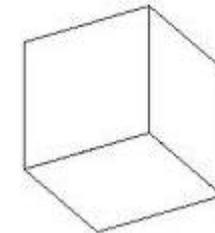
Remover faces traseiras dos objetos em relação ao observador



Adequadas para objetos convexos.

OBS :

Ser **não convexo** \neq ser **côncavo**

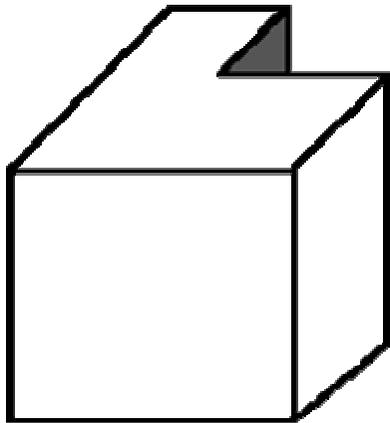


Objetos convexos

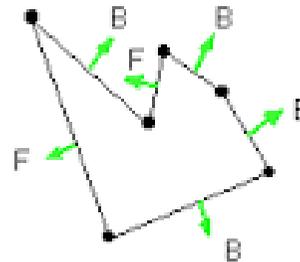
Definição:

Formado por faces convexas.

i.e. Formado por polígonos convexas: nos quais a **ligação entre quaisquer 2 pontos** internos nunca passa por uma parte externa a face:

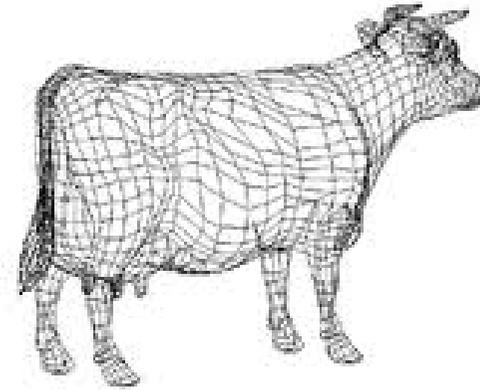


não convexo



não convexo

não convexo

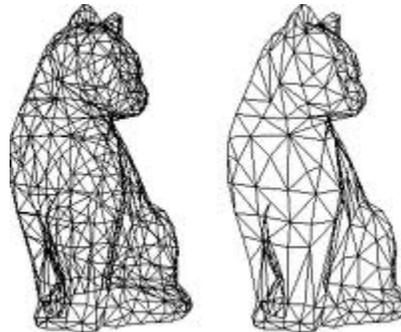


Outros exemplos:

O objeto criado pelo seu grupo é convexo?

Casa, elipse x abacaxi, ave, gato!

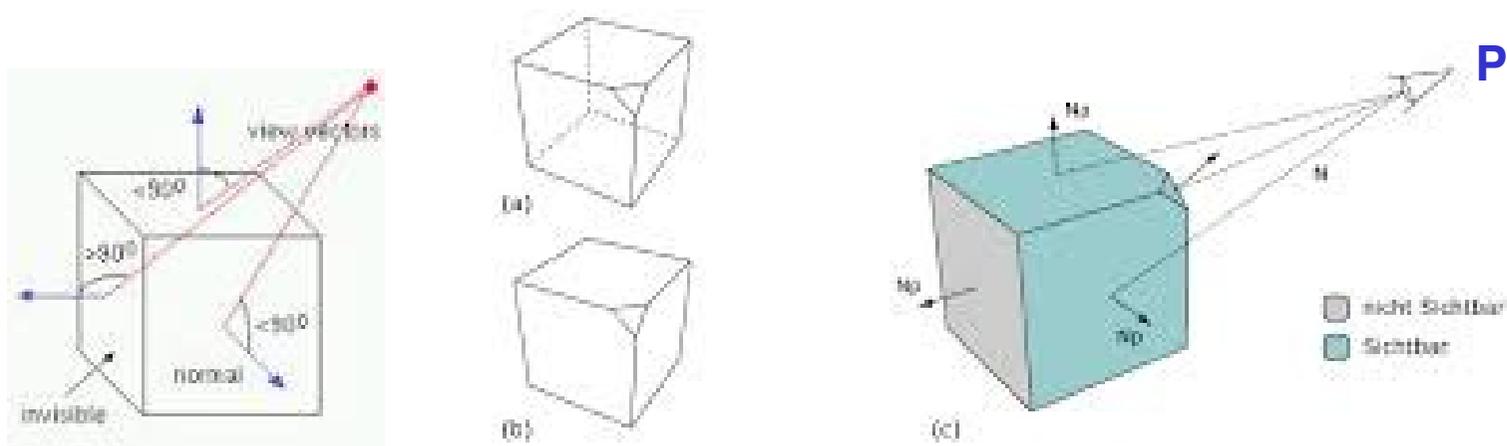
não convexo

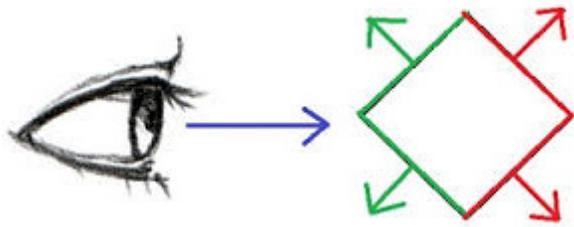


Algoritmo posiciona o objeto e o observador no mesmo sistema de coordenadas (SRU ou WC)

Usa-se a **direção que as normais** às faces fazem com a direção de visualização.

Entre **-90** graus e **90** graus a **face é visível** pelo observador (ou a face esra de frente para o observador) .

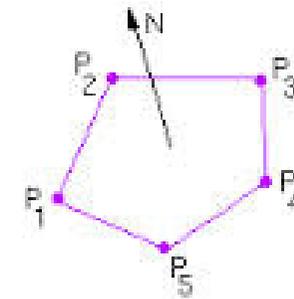
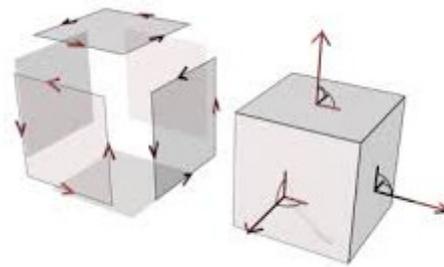




Passo 1-Obtêm a normal às faces

Sejam elas de qq numero de vértices .

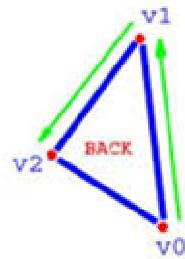
Como?



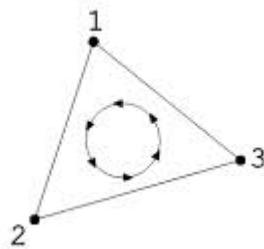
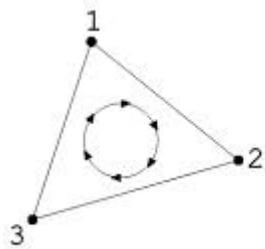
Passo 1-Obtêm a normal às faces

Como?

Através do cálculo do **produto vetorial** de dois vetores da face: Bastam 3 vertices para se ter um plano

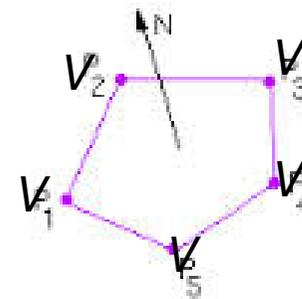


$$N = (V_1 - V_0) \times (V_2 - V_0)$$



a ordem dos vértices é importante!

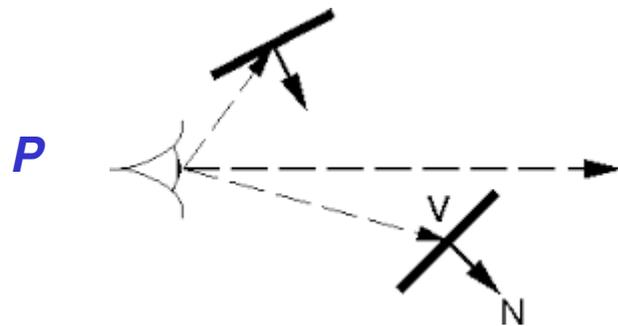
$$(V_1 - V_0) \times (V_2 - V_0) = - (V_2 - V_0) \times (V_1 - V_0)$$



Passo 2 - Define-se o vetor da direção de visão

Como?

considerando um ponto da face V_i e a posição do observador P !



Qq ponto da face V_i , por exemplo :
O primeiro V_0

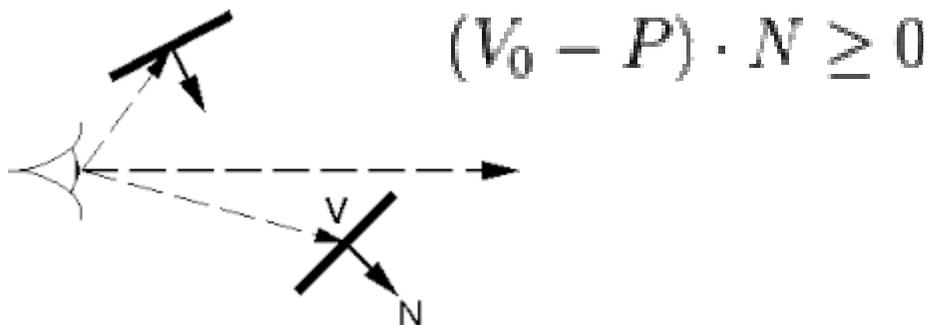
Assim $(V_0 - P)$.
É um vetor que vai do P observador a face

$$(V_0 - P) \cdot N \geq 0$$

Passo 3- Verifica-se o ângulo!

Como?

Através do **produto interno** entre as normais e a direção de visão, (não é preciso calcular o ângulo) apenas ver se o resultado **é maior que zero** → ângulo entre -90° e 90° !



Algoritmo

Passo 4- Só desenha a face se ele é visível !

OBS- Se for visível ai se preocupa em projetar o objeto de 3D para 2D e em posicioná-lo no sistema de coordenadas do dispositivo .

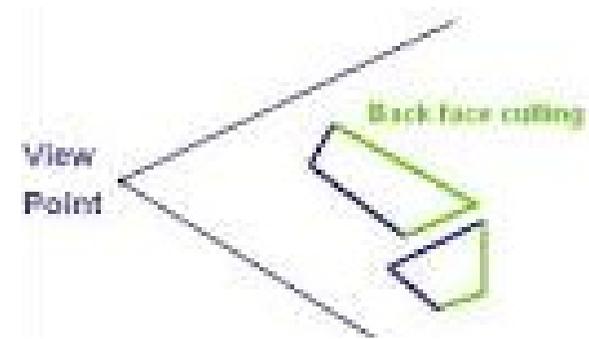
Back face culling, método de Roberts ou teste da normal

Algoritmo posiciona o **objeto** e o **observador** no mesmo sistema de coordenadas (SRU ou WC).

Não considera projeções ou perspectivas inicialmente.

Isso entra em uma outra etapa no processo de visualização (**pipeline**)

Back face culling



Demo: em javascript:

<http://echolot-1.github.io/back-face-culling-demo/>
[echolot-1/back-face-culling-demo](http://echolot-1.github.io/back-face-culling-demo/)

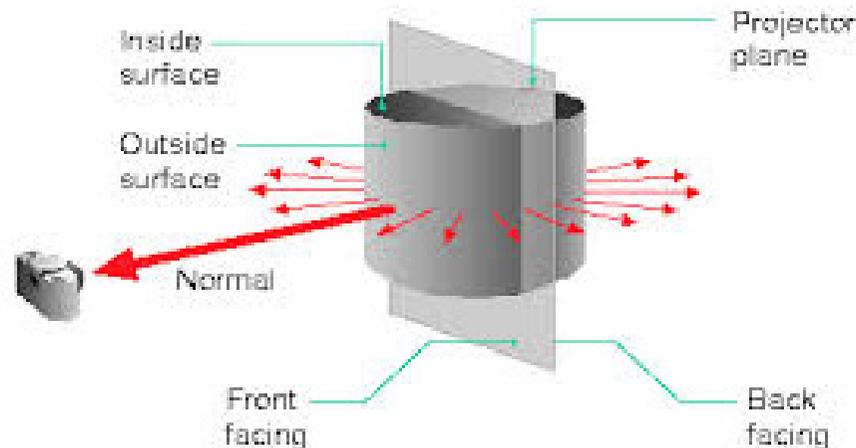
*O back-face culling determina quando **a face** de um objeto será visível de um ponto de vista.*

Esse processo torna o rendering mais eficiente pois reduz o número de polígonos a ser desenhado.

A normal é bem importante ate para a fase 2 do realismo.

Obs.

Em casos em que a superficie é definida por uma expressão, por exemplo a equação de uma curva, a normal em cada ponto pode ser obtida através desta expressão diretamente.

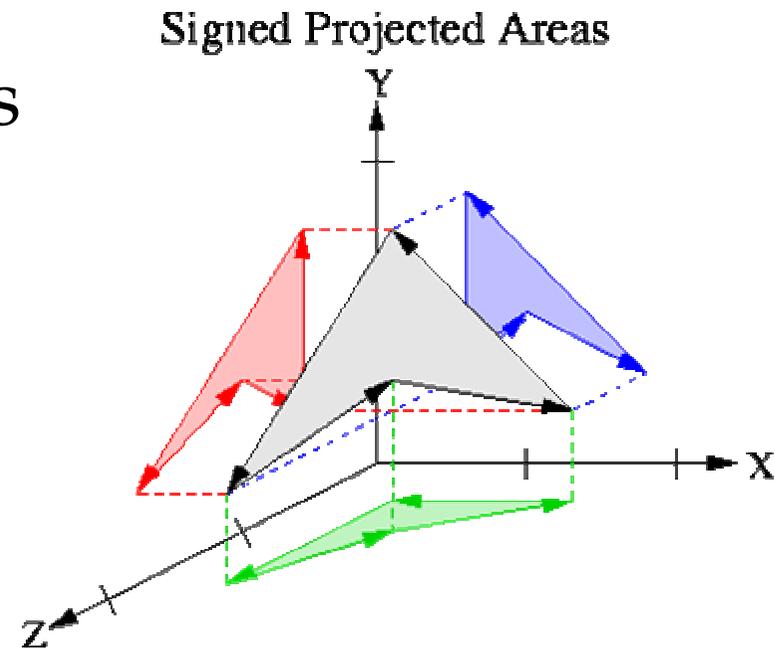


A normal é tão importante que
poderia até estar na estrutura de
dados das faces planas

Plano espacialmente
orientado no sistema de
eixos tem suas normais

Nas direções dos 3 planos
dos eixos x, y, z

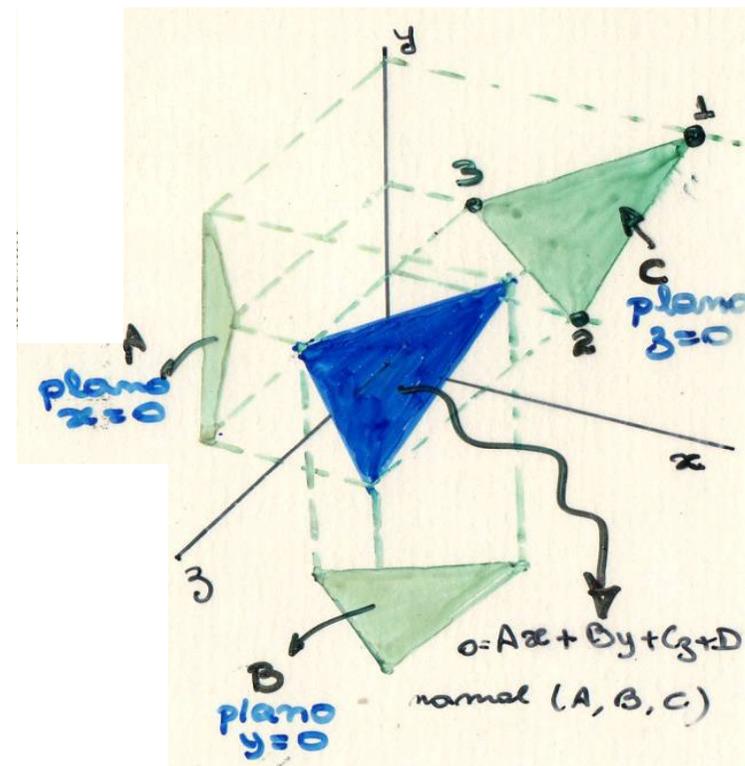
Proporcionais as áreas
projetadas nestes planos



Relações entre a normal e a equação de um plano

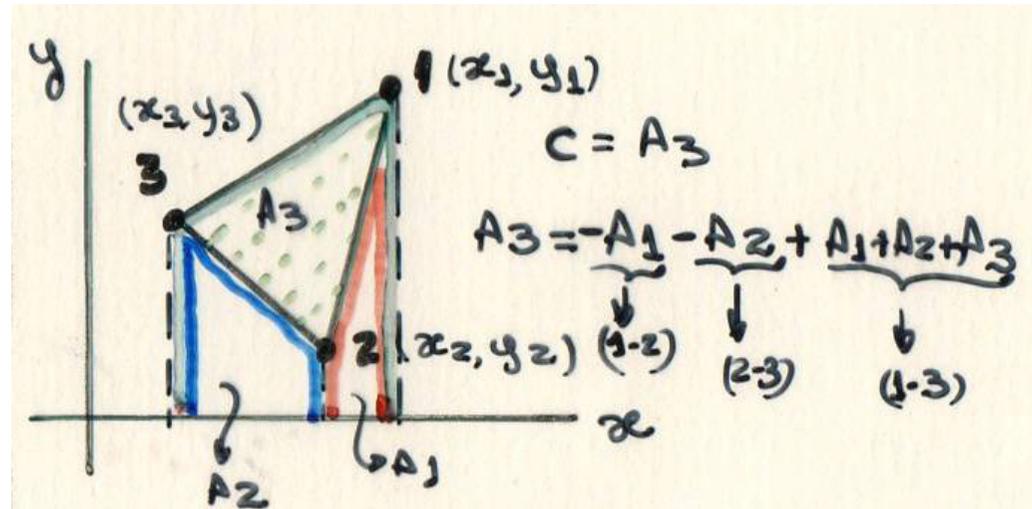
Plano no espaço tem suas normais nas direções x, y, z

Proporcionais as áreas das projeções nestes plano



Cada uma das areas projetadas

Podem ser calculadas diretamente de suas coordenadas no plano desejado, usando por exemplo o método dos trapézios:



$$C = \frac{1}{2} \sum_{i=1}^n (y_i + y_{i+1})(x_{i+1} - x_i) \quad n+1 = 1$$

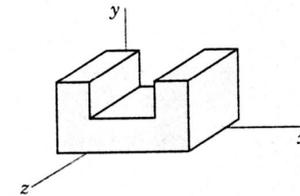
Viewing pipeline / Ações para ver uma cena

Modelagem dos objetos que compõem a Cena -SRO)

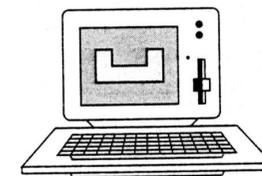
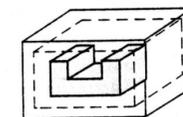
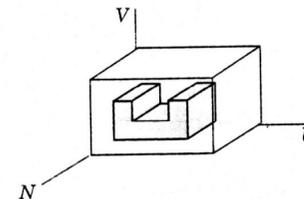
Sua posição no SRU (World Coordinates - WC), sua visão de maneira realística por um observador .

Sua vista em perspectiva e projeção em 2D , se a face for visível .

E posicionamento na window ou no canvas de desenho (DC - SRD) se a face for visível .



não convexo



Painter's algorithm - ideia

Painter's algorithm, ou **priority fill**, é uma das soluções mais simples para o problema de Visibilidade não resolvido pelo método anterior.

Na projeção de cena 3D para o plano do vídeo 2D é necessário **decidir que faces são visíveis ou escondidas** (hidden) .

O nome "painter's algorithm" se refere a técnica usada por pintores : primeiro pintam detalhes mais longes da cena e depois os cobrem com as partes mais próximas.

O **painter's algorithm** desenha os polígonos da cena **pela sua distância** ao **observador** (depth): dos mais longes para os mais próximos (**farthest to closest**).

Painter's algorithm

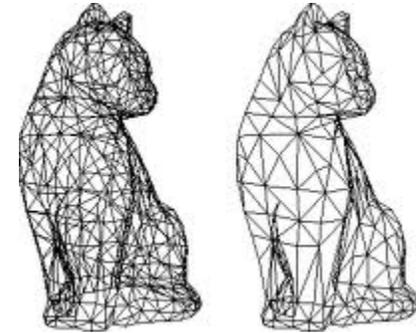
Painter's algorithm, ou **priority fill**, é uma das soluções mais simples para o problema de Visibility em 3D CG.

Continua considerando o objeto da cena 3D e o observado no mesmo cenário (hidden para um observador ou direção de vista)

Cobrindo assim as parte invisíveis – ou seja o *visibility problem* é resolvido ate para faces não convexas, com algum custo extra (o custo de ter pintado areas desnecessarias).

A ordem usada é chamada *depth order*. Essa ordenação tem uma *boa propriedade*: se um objeto obscurece parte de outro então ele é pintado depois do que vai obscurecer.

Painter's algorithm – como



Painter's algorithm se implementa: ,

Cobrindo assim as parte não mais visíveis com novas partes mais a frente :
ou seja o visibility problem é resolvido com algum custo extra baixo (the cost of having painted invisible areas).

A ordem usada é chamada ***depth order***.

Essa ordenação tem uma boa propriedade:

Algo mais a frente de outro objeto o obscurece automaticamente

RESOLVE ALGUNS CASOS DE coberturas parciais

Painter's algorithm

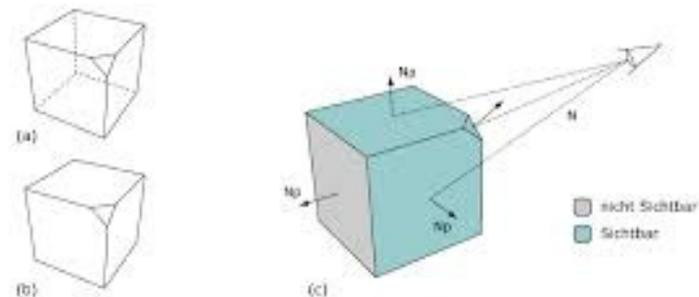
Como a distância da Face pode ser computada?

Ha muitas possibilidades:

1- *Pelo cálculo da distância média dos Vértices da Face ao observador P (X_p, Y_p, Z_p)*

2- *Fazendo uma interpolação da distancia dos vertices (V_iX, V_iY, V_iZ) ao observador P (X_p, Y_p, Z_p)*

*(V_iX, V_iY, V_iZ) - (X_p, Y_p, Z_p) para cada vertice i
Da face*

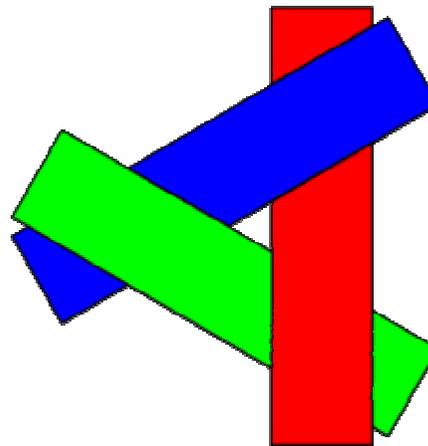


Painter's algorithm

Possibilidade de falha → quando parte MAIORES de uma face se sobrepoem a outra

→ solução divisão da face (Newell's Algorithm).

Essa falha do algoritmo e sua otimização levou ao faces cada vez merores



a distância da Face tb pode ser computada pelo
3- *Cálculo da distância de cada ponto da FACE ao observador.*

Essa é a idéia básica do **z-buffer algorithm** :
testar a distância (z - depth) de cada ponto da cena
para determinar a face mais próxima do observador
(visible surface).

Considera um array de todos os pixels a serem
pintados: $z\ buffer(x, y)$ para cada pixel (x, y) .

Esse array é inicializado com “maximum depth”.

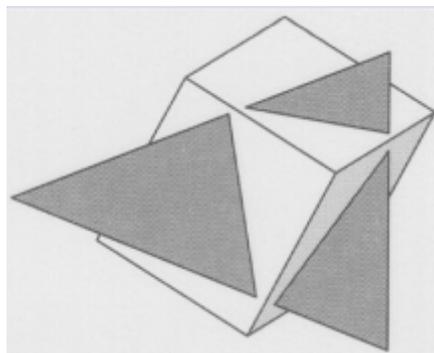
Após isso o algoritmo segue como:

Painter's algorithm

Dependendo da forma como a distância é computada, há

Possibilidade de divisão da face levou ao desenvolvimento do método de

z-buffer ou **depth buffer**



z-buffer algorithm

for cada pixel (x, y) da cena $z_buffer(x, y) = \text{maximum depth}$

for cada face P da cena

for cada pixel (x, y) de cada face P

compute z_depth de (x, y)

if $z_depth < z_buffer(x, y)$ then

set_pixel $(x, y, \text{color}) = \text{cor de } P \text{ em } (x, y)$

$z_buffer(x, y) = z_depth$

Vantagem do z-buffer:

sempre funciona e é de simples implementação!

Daqui para a frente o algoritmo considera as projeções

Isto é , supomos que já se tenha decidido como vai passar de 3D para 2D.

Qual será a direção de vista da cena!

Onde está o observador em relação aos objetos!

ALGORITMOS NA FORMA **RASTER**

(tratam pixels a pixel!)

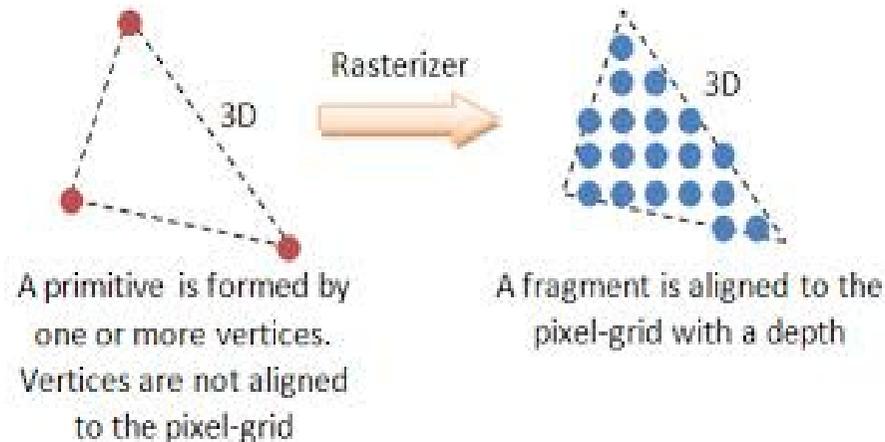
ou

semi raster (pode tratar grupo de pixels!)

Analizando os dados já projetados

ALGORITMOS NA FORMA **RASTER** ou
semi raster

(pode ser grupo de pixels e não pixel a pixel!)



ALGORITMOS NA FORMA **RASTER**

RASTER: o objeto em 3D é tratado na forma final quando já “*discretizado*” em pixels.

Rasterisation

(ou **rasterization**)

converte uma imagem descrita como **vector format** para a forma de **pixels** (dots) para representação em video, printer ou storage in a bitmap file format.

z-buffer algorithm

Idéia básica: testar a distância (z - depth) de cada superfície para determinar a mais próxima (visible surface).

Considera um array : $z \text{ buffer}(x, y)$ para cada pixel (x, y) .

Esse array é inicializado com maximum depth.
Após isso o algorithm segue como:

z-buffer algorithm

for each polygon P

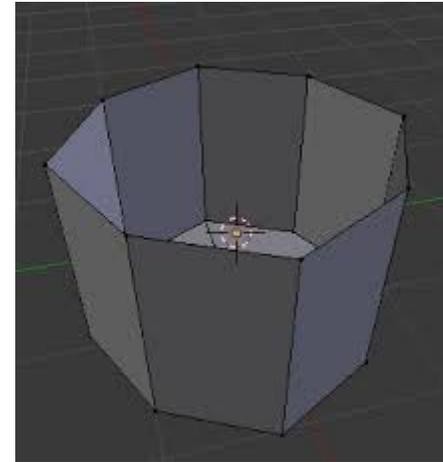
for each pixel (x, y) in P

compute z_depth at x, y

if z_depth < z_buffer (x, y) then

set_pixel (x, y, color) = intensidade de P em (x,y)

z_buffer (x, y) = z_depth



Vantagem do z-buffer:

sempre funciona e é de simples implementação!

z-buffer *algorithm*

Considerando o quando um ponto é opaco ou transparente.

Conceito de canal alfa ou composição de transparência:

Alpha compositing: processo de combinar a imagem com o fundo criando a aparência de **partial** or **full transparency**.

Idéia de translúcidos – modelo RGB α

Considere 2 polígonos, um **vermelho=R (red, 1, 0, 0, 0.5)**, e o outro **azul=B(blue, 0, 0, 1, 0.5)** renderizáveis em um fundo **verde=G(green background (0, 1, 0, 0))**.

Ambos **50% transparentes**. Se o **V(red)** estiver na frente de todos, depois o **azul (blue)** e o **verde** for o fundo (**green background**).

No final deve-se ter **50% R, 25% G** e **25% B** (Renderizando de trás para a frente as percentagens da cada cor):

Green background. (0, 1, 0)

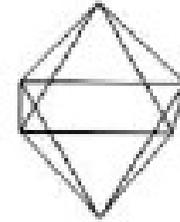
Polígono blue : (0, 0.5, 0.5) – conta 50% da cor sobre o fundo!

Polígono red: (0.5, 0.25, 0.25) – conta 50% da cor sobre outras!

z-buffer *algorithm com canal*
alfa!

ou

Alpha-blending + the Z-buffer



Given: A list of polygons $\{P_1, P_2, \dots, P_n\}$ and a background

Output: A COLOR which displays the intensity of the polygon surfaces.

Initialize: z-depth and z-buffer(x,y) , -buffer(x,y)=max depth; and
COLOR(x,y)=background at (x,y)

Begin:

z-buffer algorithm com canal alfa!

```
for(each polygon P in the polygon list)
```

```
do{
```

```
  for(each pixel(x,y) that intersects P)
```

```
  do{
```

```
    Calculate z-depth of P at (x,y)
```

```
    If (z-depth < z-buffer[x,y])
```

```
    then{
```

```
      z-buffer[x,y]=z-depth;
```

```
      COLOR(x,y)=Intensity of P at(x,y);
```

```
    }
```

```
    #considerando  $\alpha$ :
```

```
    Else if (COLOR(x,y).opacity < 100%)
```

```
    then{ COLOR(x,y)=Superimpose COLOR(x,y) in front of Intensity of P at(x,y); }
```

```
    #End consideração do  $\alpha$ :
```

```
  }
```

```
}
```

```
display COLOR array.
```



Masking Technique *ou mim Max*

Muito bom para o Hidden lines de curvas.
Isso é eliminar linhas invisíveis de superfícies

É um ALGORITMO NA FORMA **RASTER**

depende

Da direção de vista da cena!!

e

de voce já ter passado de 3D para 2D.

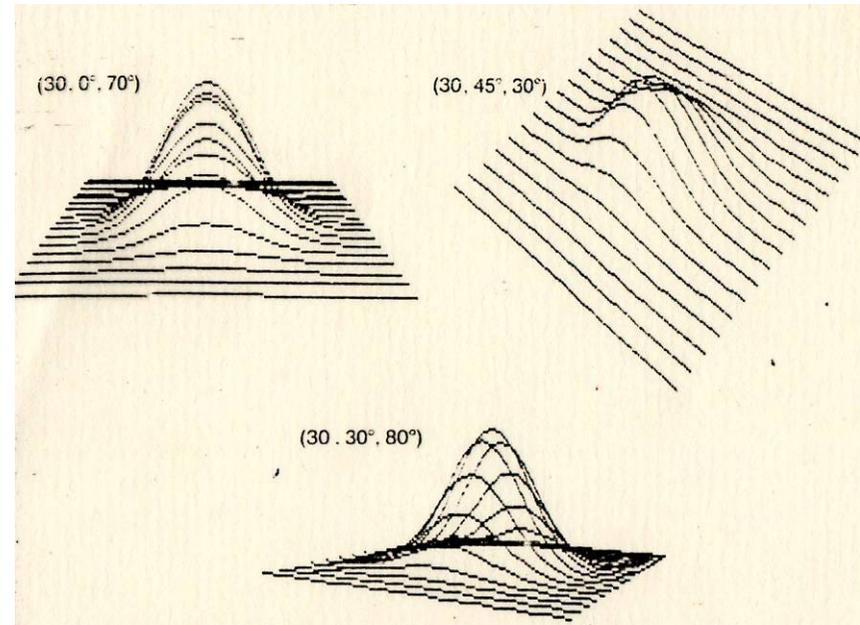
Imagine que foi gerada uma superfície

A partir de uma série de curvas.

E que voce já tem a projeção dela a partir de um certo ponto de vista.

Ou sua projeção de determinada direção

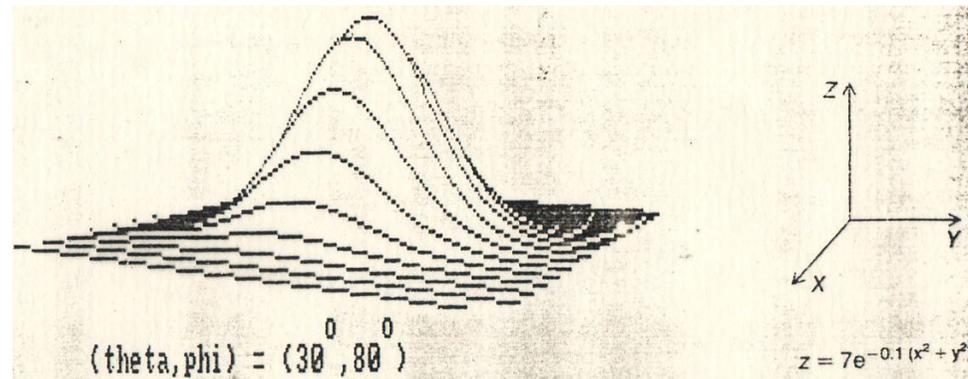
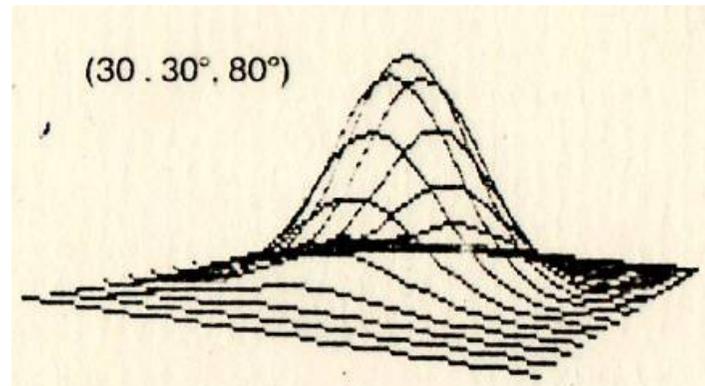
Ou seja ela já é descrita por uma série de linhas em 2D de determinada direção.



Mesma superfície representada por um conjunto de curvas e Vista de diversos pontos de vista

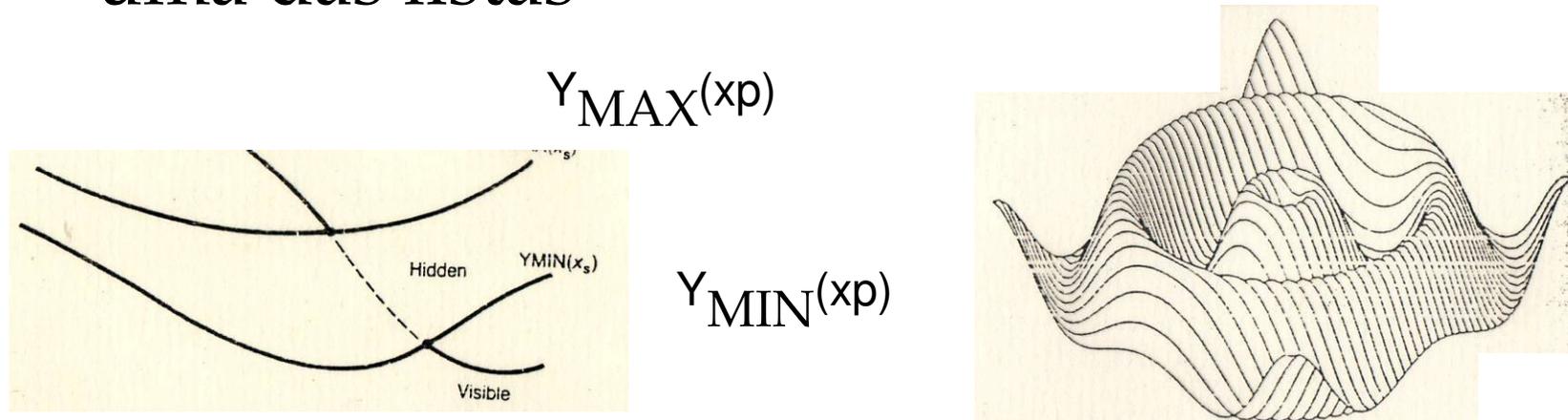
Como eliminar as linhas que são obscurecidas por partes da superfície mais a frente?

Hidden lines por mascaramento ou lista de limites verticais superiores e inferiores de cada passo (pixel) horizontal



O conceito da técnica de mascaramento

Para cada pixel ou passo de n pixels é feito 2 listas de coordenadas verticais $Y_{MAX}(xp)$ e $Y_{MIN}(xp)$ e só se desenha se algo ao ser projetado para esse xp estiver atualizando uma das listas



O número de pixel usado, ou o passo

Pode ser uma função da curvatura da superfície ou curva.

Mais curvatura menor passo!!

E como se obtém a curvatura?

O que é curvatura?

curvatura

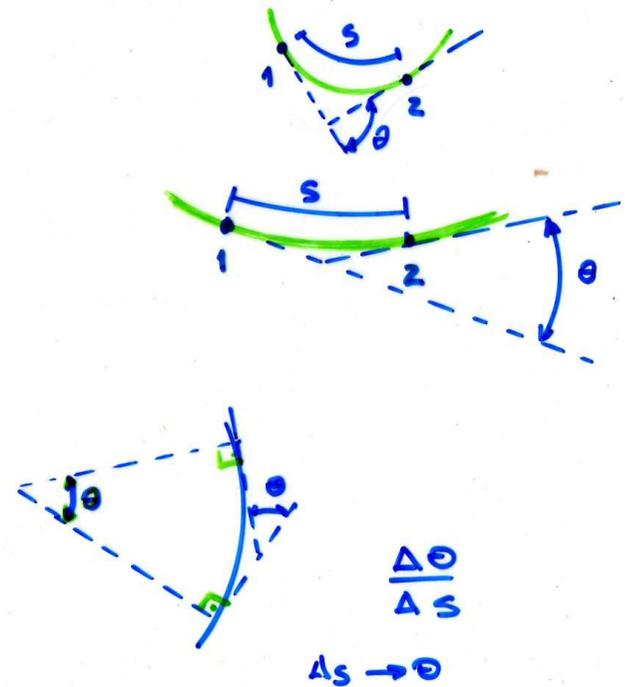
derivada do ângulo formado por 2 Tangentes à curva em relação ao comprimento do arco entre essas 2 Tangentes

$$\frac{d\theta}{ds}$$

em um círculo $ds = R d\theta$

logo curvatura $\frac{1}{R}$

para os círculos



modelo de realismo

utilizado para calcular como se renderiza um objeto podem ser:

Locais x Globais

Em WC ou em DC

Modelos globais

Ao contrario dos modelos locais que consideram a superfície a luz e o observador, os globais consideram todos os objetos da cena, precisam ter toda a base de dados dos objetos

Principais: Raytracing e radiossidade

Não produzem os mesmo efeitos nem são adequados pra as mesmas coisas!

Lentos para real time!

Ray tracing *simplificado ou aproximado* *ou*

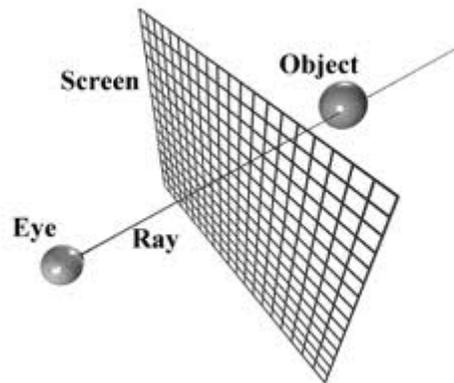
Ray casting lança raios a partir do observador de forma a perceber a distância dos objetos que compõem a cena.

Os raios são emitidos **a partir do observador**, (no sentido inverso do que acontece na natureza), para reduzir recursos computacionais (pois a maior parte dos raios de luz que partem da fonte não chegam ao observador).

Ray casting

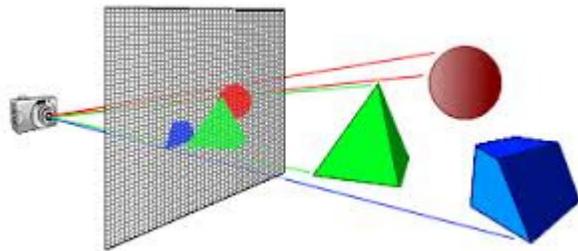
Supõe-se um raio do olho do observador passando por **cada ponto da tela** a ser desenhada. O ponto da tela receberá a cor do objeto que for atingido na cena pelo raio.

O calculo das interseções é o ponto chave do algoritmo.



Ray casting

permite remover as superfícies escondidas utilizando as informações obtidas a partir das primeiras intersecções encontradas pelos raios lançados a partir do observador.



Ray tracing (rastreamento)

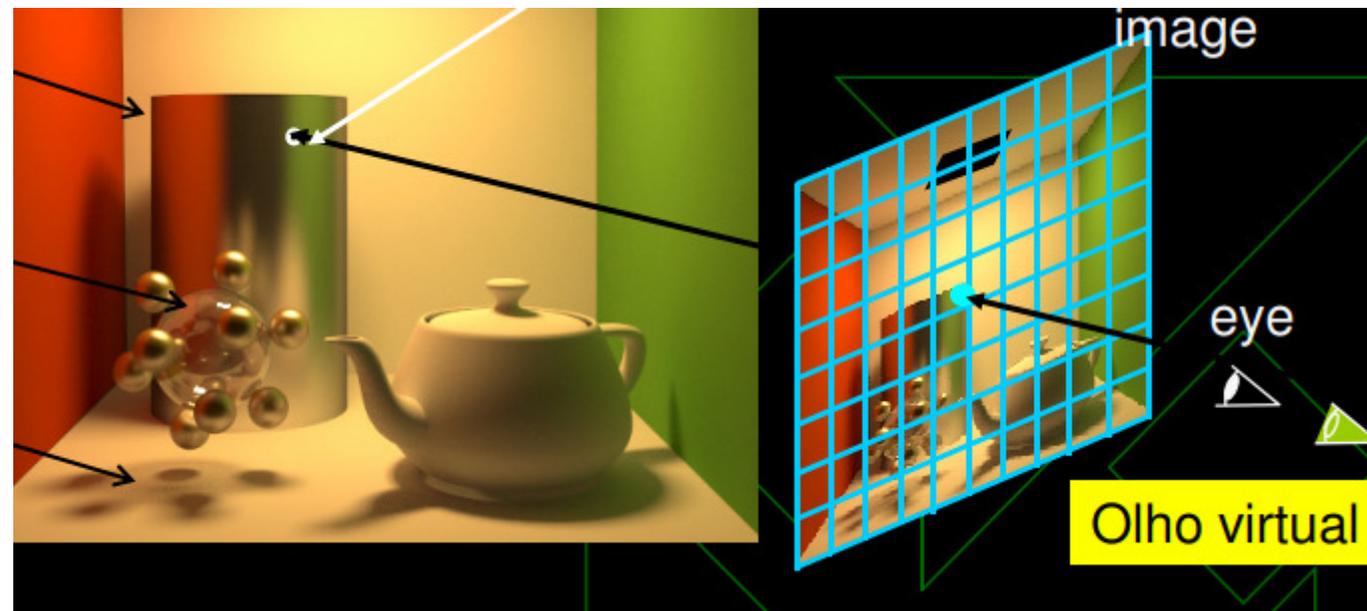
Método recursivo, onde recorre ao lançamento de raios secundários a partir das interseções dos raios primários com os objetos.

Ray casting é apropriado para a renderização de jogos 3D em tempo-real.

Durante a viagem do raio pode acontecer: absorção, reflexão ou refração. A superfície pode refletir toda ou apenas uma parte do raio numa ou mais direção. A soma das componentes absorvidas, refletidas e refratadas tem que ser igual ao inicial.

Duvida em como calcular as

Interseções?



Cálculo de interseções:

A tarefa principal do ray tracing consiste no cálculo da interseção de um raio com o objeto. Para essa tarefa, utiliza-se normalmente a representação paramétrica de um vetor ou reta. Cada ponto (x, y, z) ao longo de um raio com origem no ponto (x_0, y_0, z_0) e direção do ponto (x_0, y_0, z_0) para o ponto (x_1, y_1, z_1) é definido em função do parâmetro t , (com valores no intervalo $[0,1]$ pelas equações paramétricas da reta):

$$x = x_0 + t(x_1 - x_0);$$

$$y = y_0 + t(y_1 - y_0);$$

$$z = z_0 + t(z_1 - z_0);$$

$$x = x_0 + t\Delta x; \Delta x = x_1 - x_0$$

$$y = y_0 + t\Delta y; \Delta y = y_1 - y_0$$

$$z = z_0 + t\Delta z; \Delta z = z_1 - z_0$$

(x_0, y_0, z_0) for considerado o centro de projeção, ou o olho do observador

(x_1, y_1, z_1) for o centro de um pixel na “janela”

t varia de 0 a 1 entre esses pontos.

valores de t maiores que 1 correspondem a pontos depois da janela

Raytracing

Bom para:

reflexões,

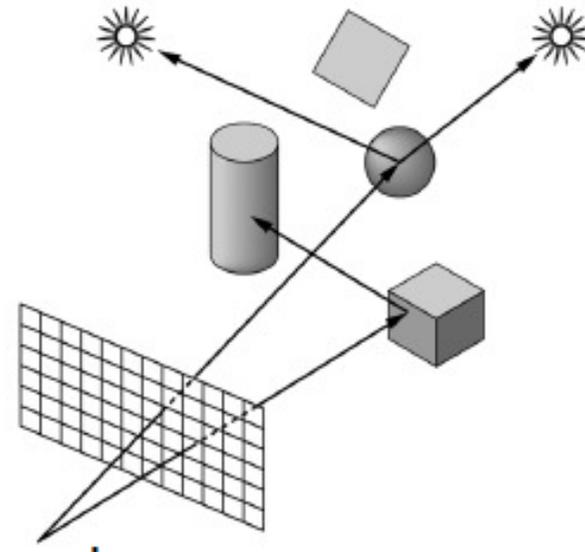
transparências,

objetos fáceis de
calcular interseções

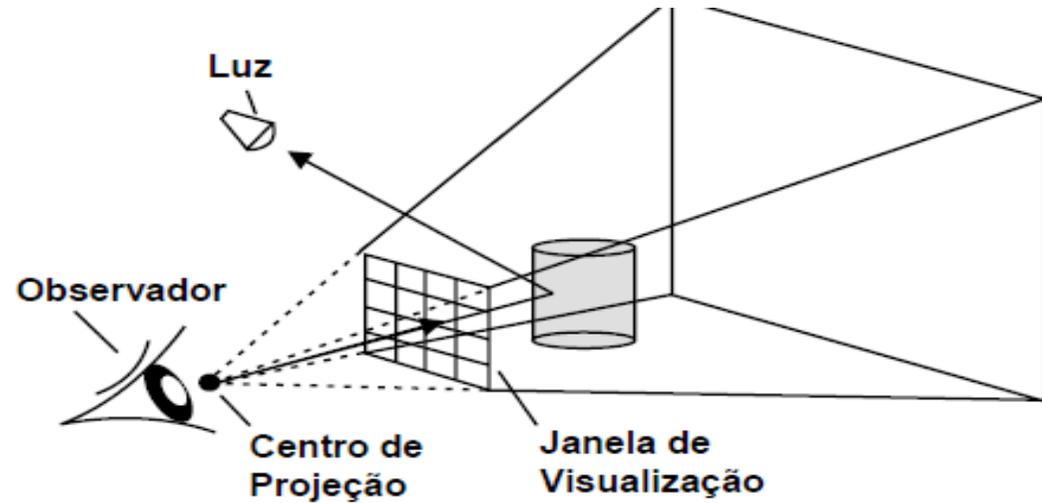
(superfícies,

planas, esférica,

E cilíndricas)



Ray tracing



é uma técnica para gerar uma imagem, seguindo o caminho da luz através de pixels em um plano de imagem e simulando os efeitos de seus encontros com objetos.

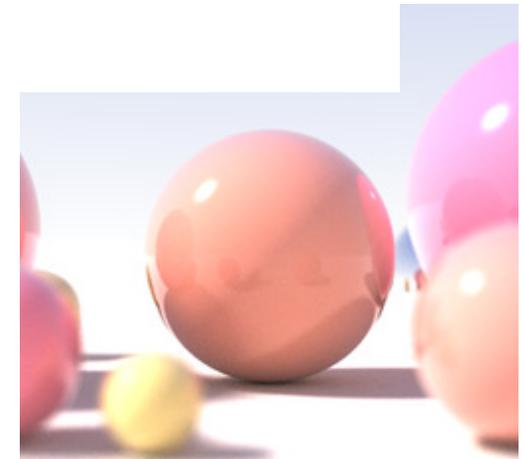
é capaz de produzir um elevado realismo visual, geralmente maior do que o dos métodos de processamento locais típicos, mas em um maior custo computacional.

Isso faz com ray tracing mais adequado para aplicações em que a imagem pode ser renderizada lentamente, como em imagens de cinema e televisão, efeitos visuais, e pouco adequada para aplicações em tempo real, como jogos, onde a velocidade é fundamental.

é capaz de simular uma variedade de efeitos ópticos, como os fenômenos de dispersão, reflexão e refração.

O algoritmo de Ray tracing considera os seguintes pontos:

- Os raios são disparados de forma sistemática, de modo que cada um deles corresponda a um pixel na tela.
- Após o disparo, o raio percorre o espaço podendo atingir um objeto ou sair da cena.
- Se atingir algum objeto, o ponto de intersecção é calculado. As contribuições das fontes de luz para cada ponto, levando em conta a sombra de outros objetos, também são calculadas.
- Se o objeto for opaco, a soma dessas contribuições será a intensidade luminosa total naquele ponto.
- Caso contrário, as contribuições devidas aos reflexos e refrações, serão também computadas. O pixel correspondente pode, então, ser exibido.
- Se não houver intersecção, o pixel terá a cor de fundo.



Algoritmo clássico

Para cada pixel da tela:

1. Trace um “raio” a partir do observador até a cena a ser representada através de um pixel da tela;
2. Determine qual o primeiro objeto a interceptar esse raio;
3. Calcule a cor ambiente da superfície do objeto no ponto de interseção baseado nas características do objeto e na luz ambiente;
4. Se a superfície do objeto for reflexiva, calcule um novo raio a partir do ponto de interseção e na “direção de reflexão”;
5. Se a superfície do objeto for transparente, calcule um novo raio a partir do ponto de interseção.
6. Considere a cor de todos os objetos interceptados pelo raio até sair da cena ou atingir uma fonte de luz, e use esse valor para determinar a cor do pixel e se há sombras.



Espelhos:

O ray tracing deve considerar os raios refletidos toda vez que o coeficiente de reflexão de uma superfície for diferente de zero. O coeficiente de reflexão varia entre 0 e 1, determinando que quantidade de energia do raio de luz deve ser considerada como absorvida pelo objeto em questão, compondo uma soma ponderada das componentes de cor para o pixel na tela. Um espelho possui um coeficiente de reflexão próximo de 1, ou seja, nessa superfície todos os raios incidentes devem ser refletidos com o mesmo ângulo de incidência em relação à direção da reta normal à superfície. Além do coeficiente de reflexão, as superfícies também apresentam um coeficiente de refração que expressa a maneira pela qual a luz passa através de um meio para outro.

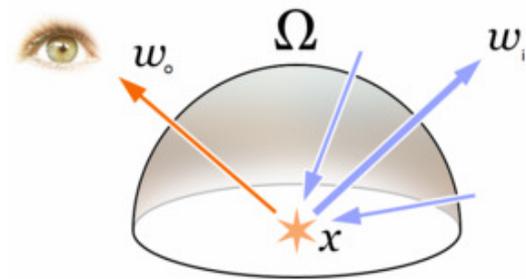


Radiosidade

considera a solução da integral de rendering (equilíbrio da radiância em um ponto ou a conservação da energia) para modelar a iluminação.

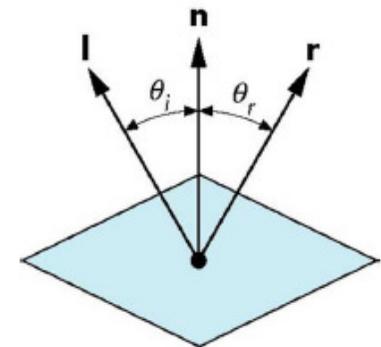
O nível de realismo da modelagem é muito maior.

Considera a função bidirecional de distribuição da reflectancia-
bidirectional reflectance distribution function (BRDF).

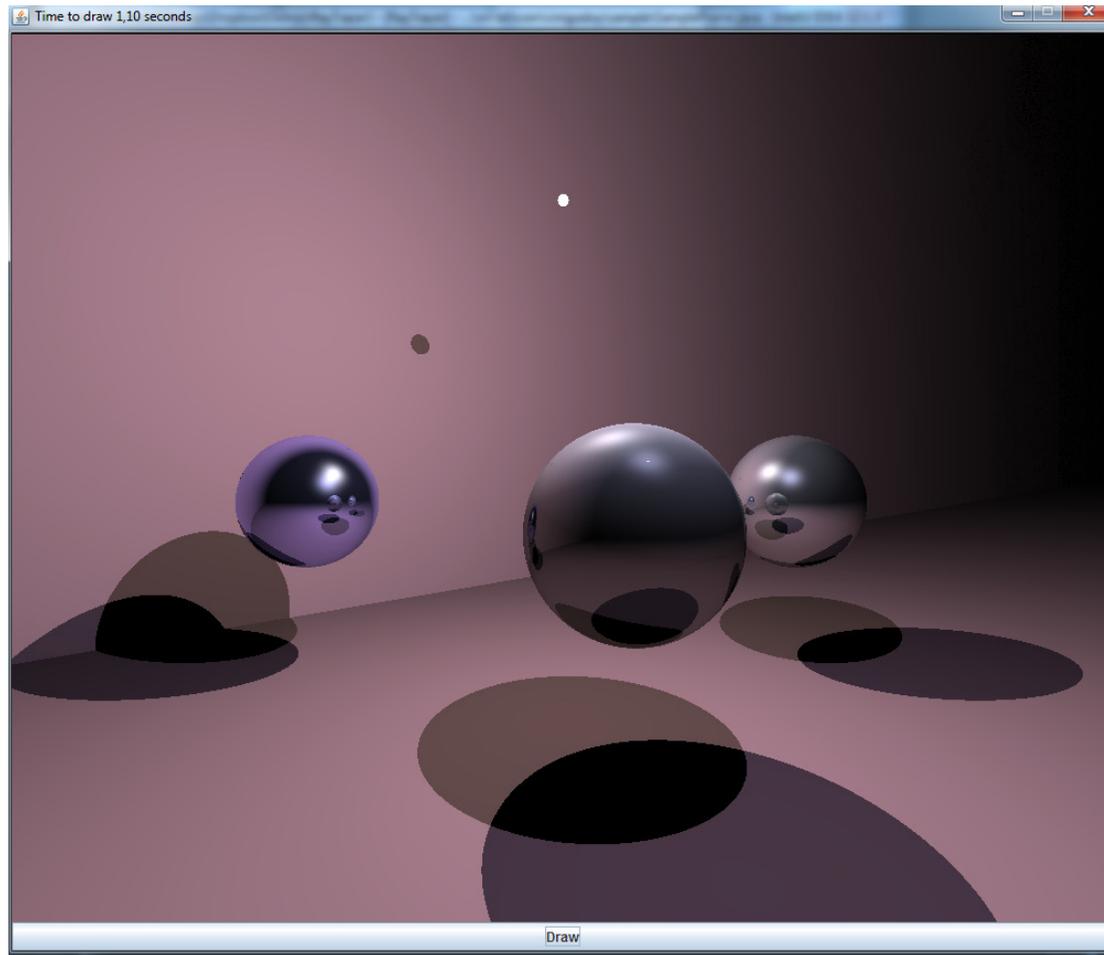


Os anteriores todos consideram
Que os 3 vetores estão no mesmo plano
(reflexão ideal)

$$\mathbf{r} = 2 (\mathbf{l} \cdot \mathbf{n}) \mathbf{n} - \mathbf{l}$$



Cena calculada por ray tracing



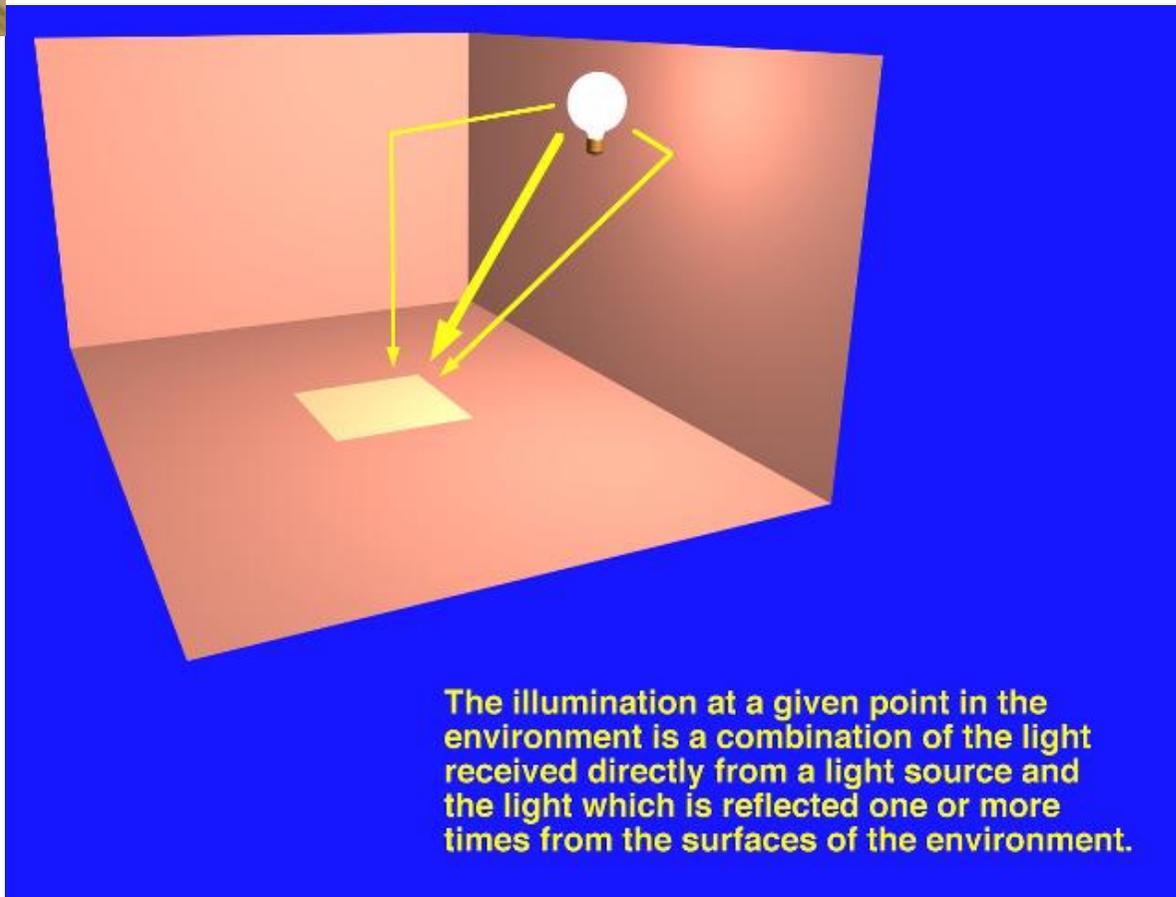
Radiosidade é:

- uma aplicação do método de elementos finitos para resolver a equação de renderização para cenas com superfícies que refletem a luz de forma difusa.
- um algoritmo de iluminação global: a iluminação não vem apenas a partir das fontes de luz, mas todas as superfícies de cena interagindo uns com os outros.
- independente do ponto de vista**, o que aumenta o volume dos cálculos envolvidos, mas torna-os **úteis para todos os pontos de vista**.
- inicialmente uma aplicação desenvolvidos na área de transferência de calor, posteriormente adaptada para a aplicação de computação gráfica (1984 na Universidade de Cornell).

Radiosidade: discretiza o ambiente em um malha

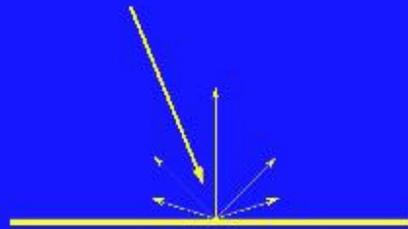


Os limites da malha devem coincidir com os limites das zonas de diferença de iluminação

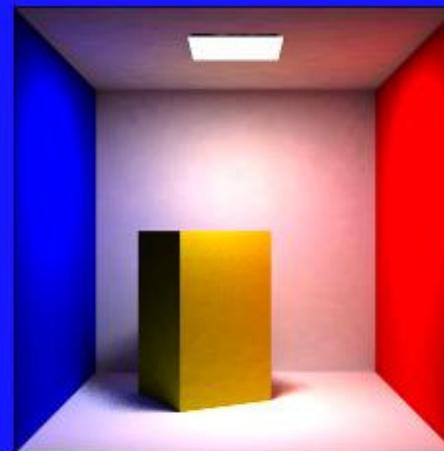


The illumination at a given point in the environment is a combination of the light received directly from a light source and the light which is reflected one or more times from the surfaces of the environment.

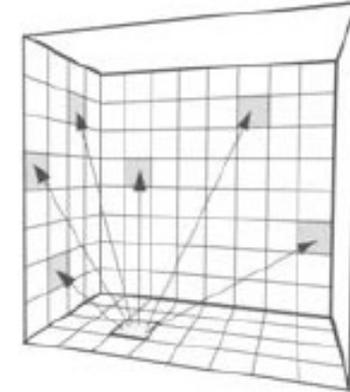
Balanço ou equilíbrio de energia radiante



Light striking a surface is reflected in all directions, following the Lambertian reflection model. This diffuse reflection of light leads to color bleeding, as light striking a surface carries that surface's color into the environment.



Radiosidade:



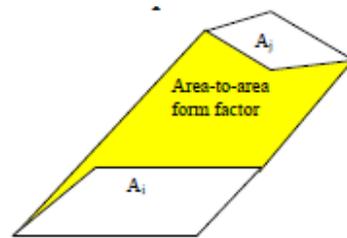
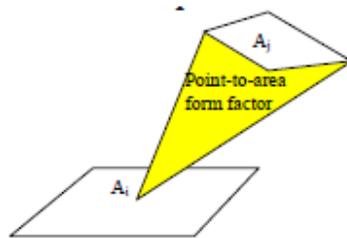
O método da radiosidade é baseado em um modelo simples de balanço de energia. Na sua origem, o cálculo da radiosidade empregado em Transmissão de Calor não é mais do que a aplicação da lei da conservação da energia a cada uma das superfícies de um recinto ou cena, e pressupõe a existência de equilíbrio térmico. Em cada superfície de um modelo, a quantidade de energia emitida é a soma entre a energia que a superfície emite internamente mais a quantidade de energia refletida. A quantidade de energia refletida pode ser caracterizada pelo produto entre a quantidade de energia incidente na superfície e a constante de reflexão da superfície.

$$B_j = \rho_j H_j + E_j$$

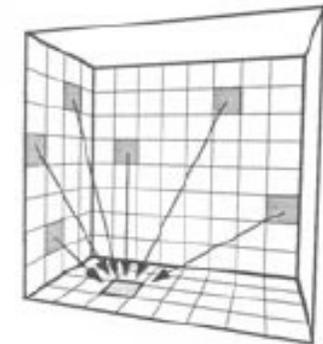
onde B_j é a radiosidade da superfície j , ρ_j sua reflectividade, H_j a energia incidente nesta superfície e E_j a energia emitida pela superfície j

Radiosidade

A radiosidade de uma superfície é a energia dissipada. Isso é usado para determinar a intensidade luminosa da superfície. A quantidade de energia emitida por uma superfície deve ser especificada como um parâmetro do modelo, como nos métodos tradicionais onde a localização e a intensidade das fontes de luz devem ser especificadas. A reflectividade da superfície também deve ser especificada no modelo, como nos métodos de iluminação tradicional. A única incógnita da equação é a quantidade de luz incidente na superfície. Esta pode ser encontrado somando-se todas as outras superfícies à quantidade de energia refletida que contribui com a iluminação dessa superfície:



$$H_j = \sum_{i=1}^n B_j F_{ij}$$



onde H_j é a energia incidente na superfície j , B_j a radiosidade de cada superfície i da cena e F_{ij} uma constante $i j$.

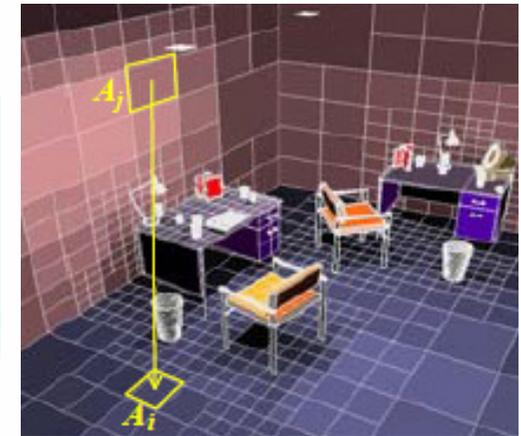
A constante dessa equação é definida como a fração de energia que sai da superfície i e chega na superfície j , e é, portanto, um número entre 0 e 1. Essa constante pode ser calculada por métodos analíticos, ou através de semelhança geométrica.

A equação da radiosidade fica assim:

$$B_j = E_j + \rho_j \sum_{i=1}^n B_i F_{ij}$$

A consideração de todas as superfícies da cena forma uma seqüência de N equações lineares com N incógnitas, o que leva a uma solução matricial:

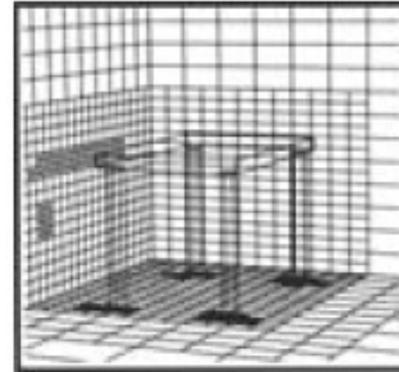
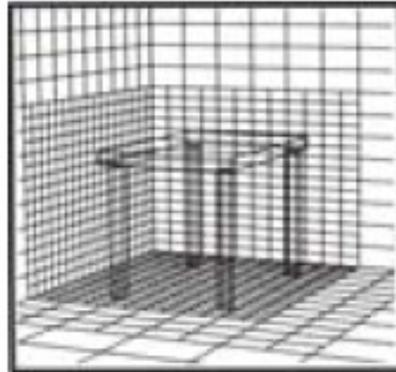
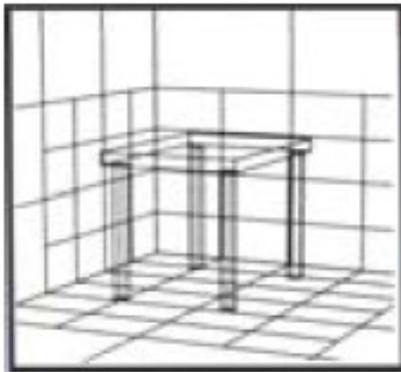
$$\begin{bmatrix} 1 - \rho_1 F_{11} & 1 - \rho_1 F_{12} & \Lambda & -\rho_1 F_{1n} \\ -\rho_2 F_{21} & 1 - \rho_2 F_{22} & \Lambda & -\rho_2 F_{2n} \\ M & M & O & M \\ -\rho_n F_{n1} & -\rho_n F_{n2} & \Lambda & -\rho_n F_{nn} \end{bmatrix} \begin{bmatrix} B_1 \\ B_2 \\ M \\ B_3 \end{bmatrix} = \begin{bmatrix} E_1 \\ E_2 \\ M \\ E_3 \end{bmatrix}$$



Essa matriz tem duas propriedades interessantes: é diagonalmente dominante e, portanto, converge quando usada no método iterativo de Gauss Seidel, [GOR, 84]. Métodos alternativos para o cálculo dessa matriz já foram propostos por Cohen et al. [COH, 88]. Alguns permitem uma convergência para a solução correta mais rápida que o algoritmo iterativo de Gauss Seidel.

Refinamentos progressivos

Alterando o numero de elementos da malha:



Coarse patch solution
(145 patches)



Improved solution
(1021 subpatches)



Adaptive subdivision
(1306 subpatches)

Color bleeding

Em rendering , **color bleeding** é a ocorrência de colorização de um objeto ou superfície pela cor refletida de superfícies próximas.

Ocorre principalmente quando se usa Radiosity para a cena 3D.



Photon mapping

Algoritmo de **iluminação global** em 2 passadas (two-pass) que considera modelos de radiância para maior realismo na simulação da refração e reflexão da luz em superfícies transparentes

É capaz de simular a refração da luz em meios transparentes tal como o vidro ou a água, interreflexões difusas entre objetos iluminados, a dispersão da luz sob a superfícies de materiais translúcidos, e efeitos causados por partículas, tal como o **fumaça ou a água de vapor**.

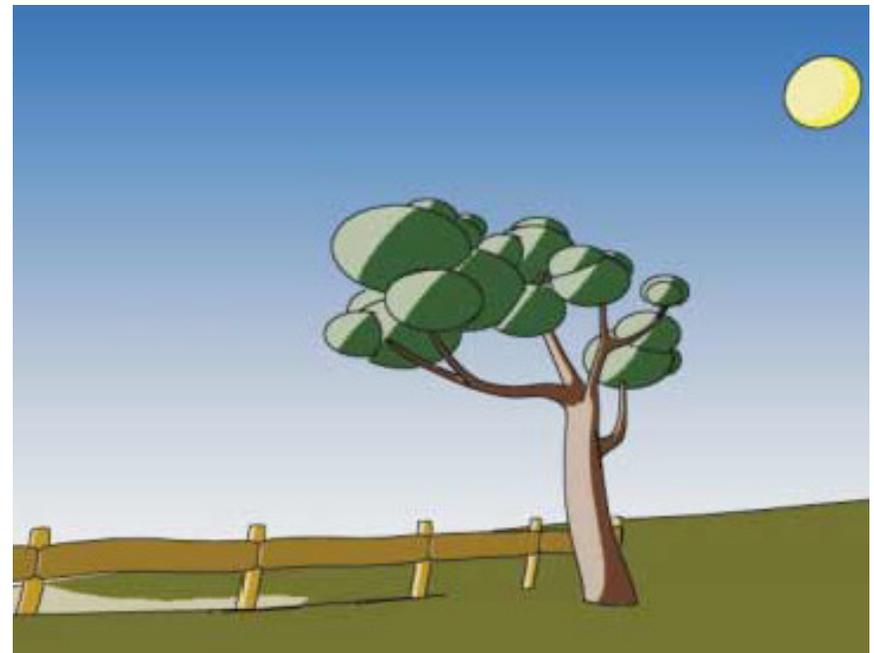
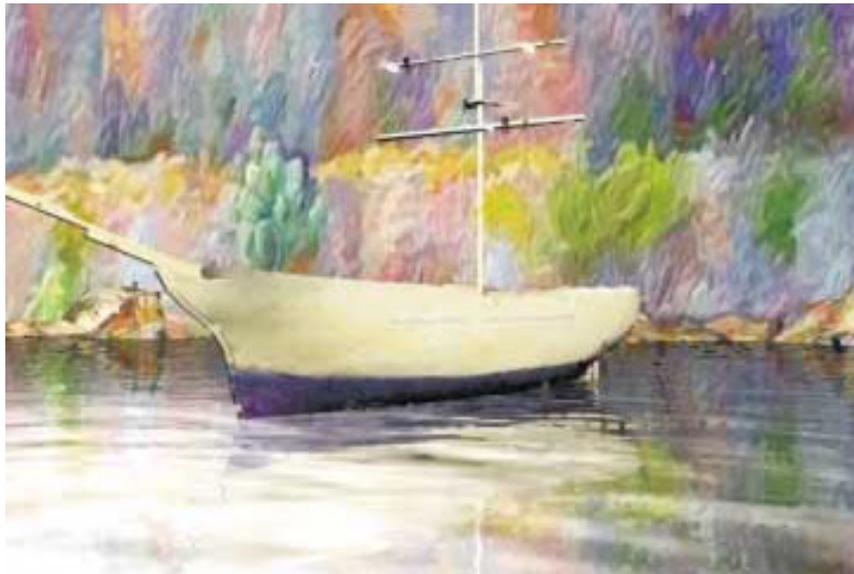


Há muito mais do que isso!

Vimos aqui apenas sobre um realismo fotográfico das imagens, mas há diversas outras formas e esse assunto esta sempre em constante evolução. Assim depois desta leve introdução continue na área! Você já tem a bagagem teórica que precisa para agora descobrir o resto sozinho!

Toon Shading

Stylistic rendering



Bibliografia:

D. F. Rogers, J. A. Adams. Mathematical Elements for Computer Graphics, 2dn Ed. , Mc Graw Hill, 1990

E. Azevedo, A. Conci, [Computação Gráfica: teoria e prática](#), [Campus](#) ; - Rio de Janeiro, 2003

J.D.Foley,A.van Dam,S.K.Feiner,J.F.Hughes. Computer Graphics- Principles and Practice, Addison-Wesley, Reading, 1990.

Y. Gardan. Numerical Methods for CAD , MIT press, Cambridge, 1985.

A. H. Watt, F. Policarpo - [The Computer Image](#) , Addison-Wesley Pub Co (Net); 1998

https://noppa.oulu.fi/noppa/kurssi/521493s/luennot/521493S_3-d_graphics_vi.pdf

<http://graphics.stanford.edu/papers/rad/>