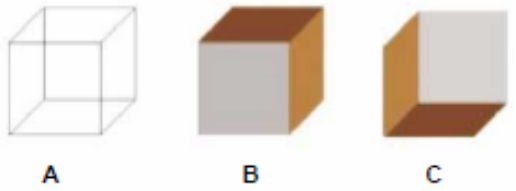
**aula 14**
**Fases do Realismo Visual**
**IC/UFF - 2016**

# Fases do realismo




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

Geometria dos objetos da cena

Representação 3D (wire frame)

Eliminação de partes não visíveis

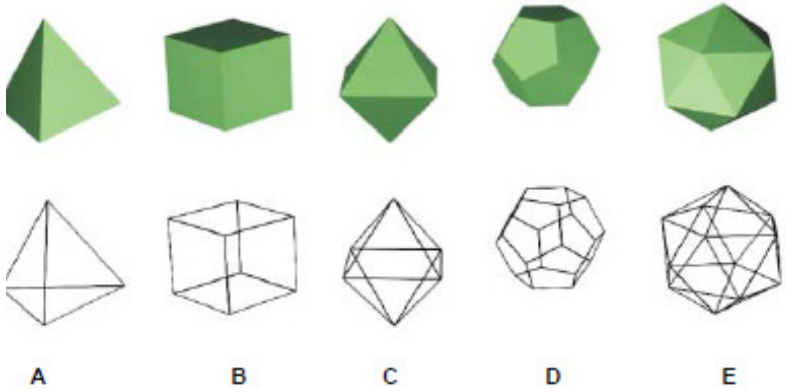
Shading (reflexão difusa)

Iluminação (reflexão  especular)

Sombras (shadow)

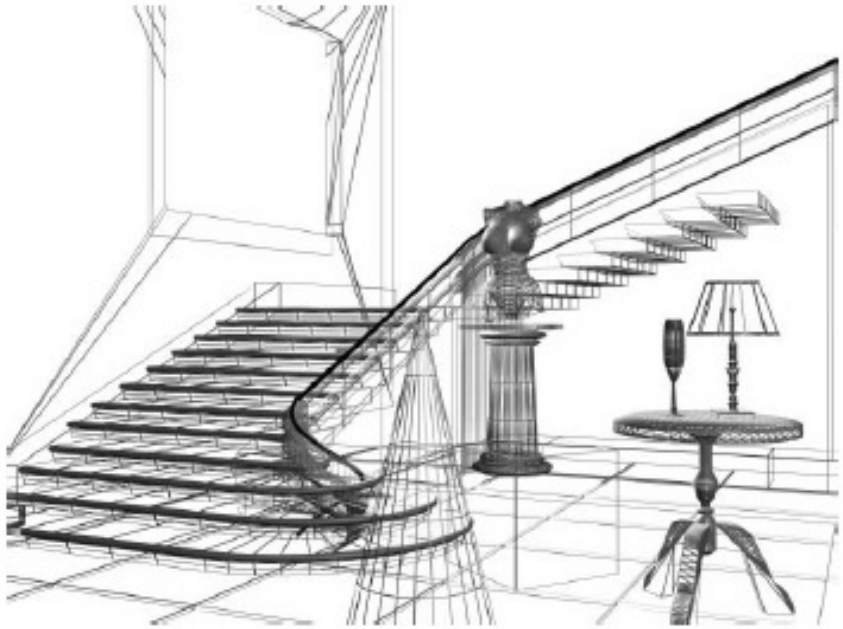
Reflexão, transparências, refração

Texturas




*Os 5 tipos possíveis de poliédros regulares.*

# Wire frame (modelagem, projeção, volume de visão)



*Representação dos objetos em wire-frame.*

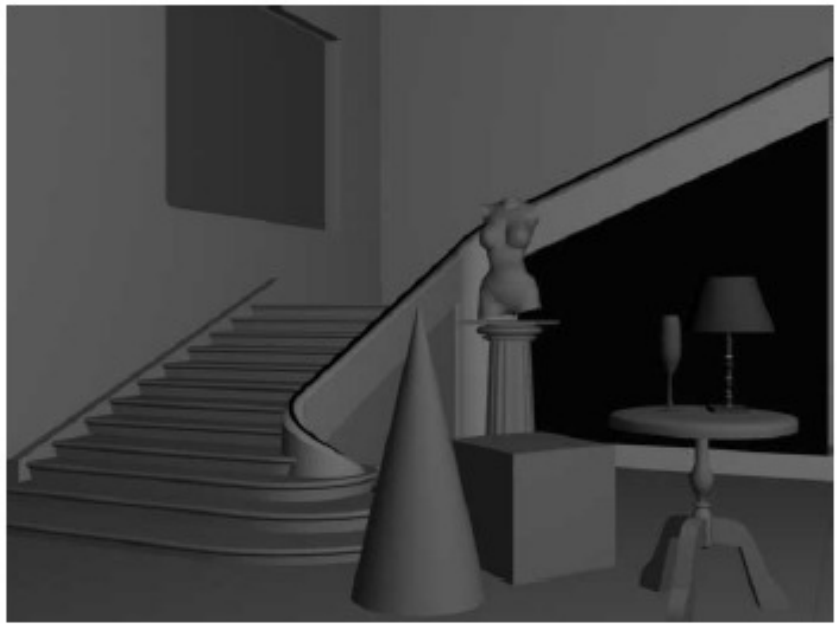
# Tratamento de hiddens

*Remoção das linhas não-visíveis.*

# Shading com luz ambiente

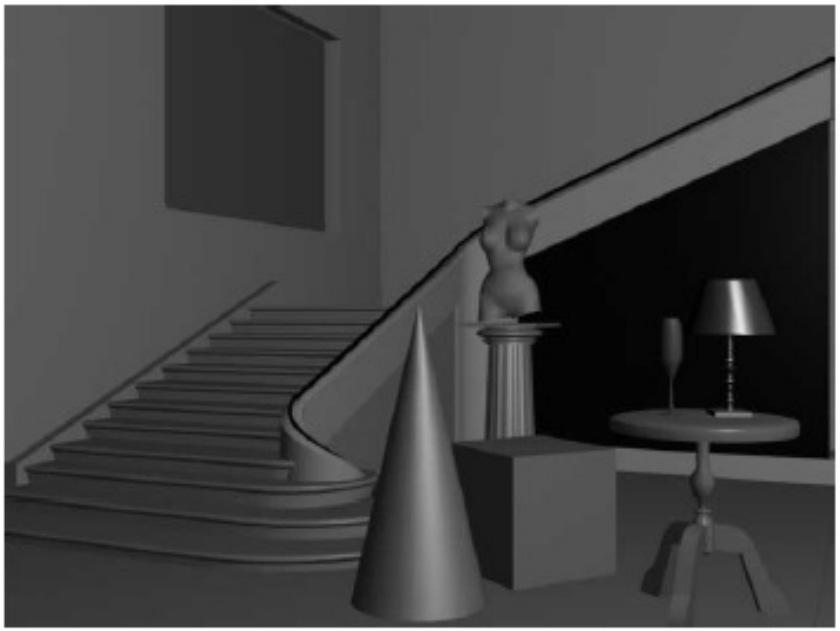
*Reflexão Ambiente aplicada em uma cena.*

# Luz direcional



*Reflexão Difusa aplicada em uma cena.*

# Tratamento de Iluminação especular



*Reflexão Especular aplicada em uma cena.*

# Cores e texturas

# Sombras e reflexão

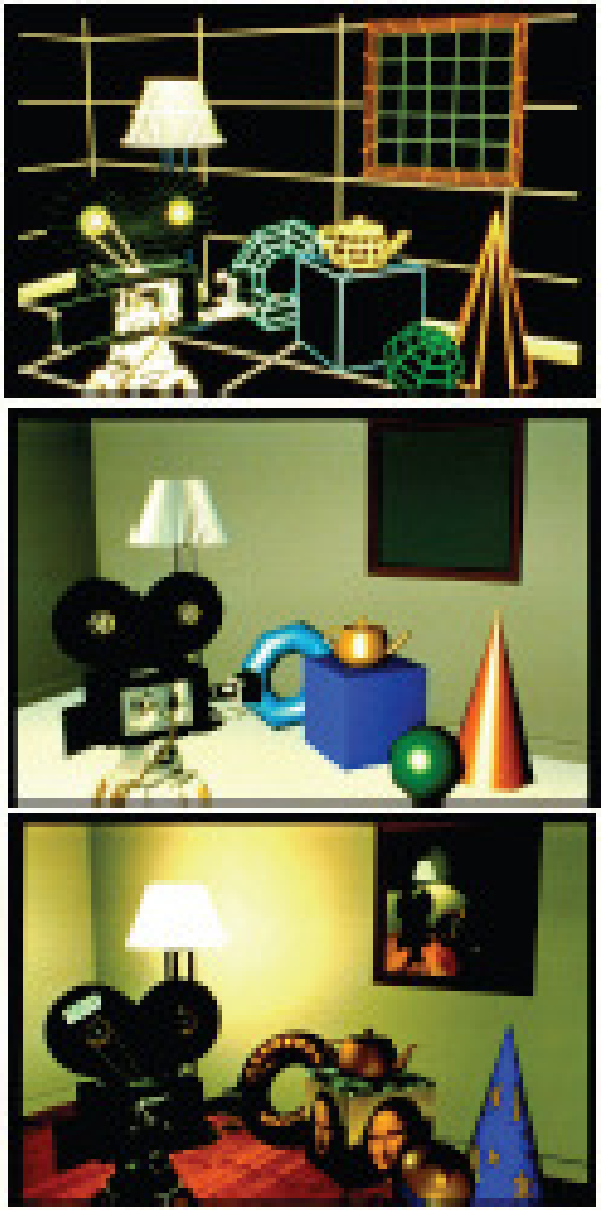
# Realismo em passos

## Objetivos

Melhorar o entendimento das cenas e objetos criados

Possibilidade de representação de dados, objetos e cenas complexas

Realismo até o nível desejado da forma adequada para a aplicação (real time x perfeição física da cena)



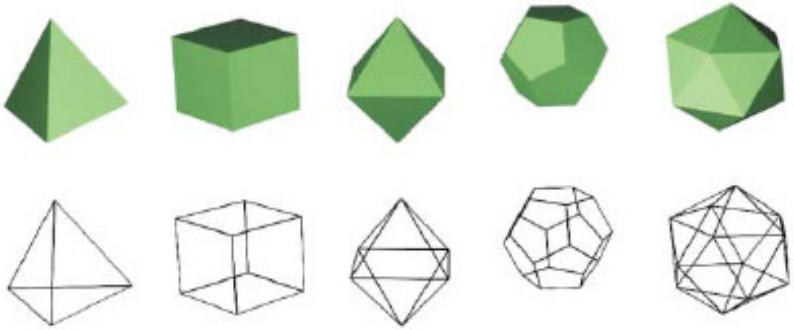
# Nível adequado do realismo

Remoção de partes invisíveis do objeto (linhas, superfícies e oclusões por outros objetos)

Sombreamento das diversas superfícies ou *Shading* :

reflexão difusa,

reflexão especular



Demais níveis de detalhes:

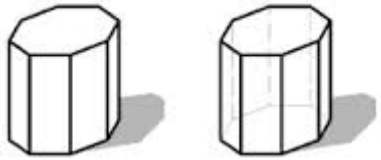
Sombras (*shadows*)

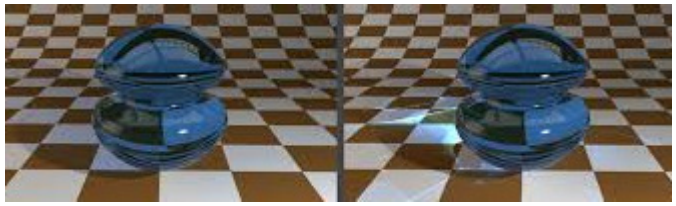
Reflexão,

transparências,

refração,

Texturas

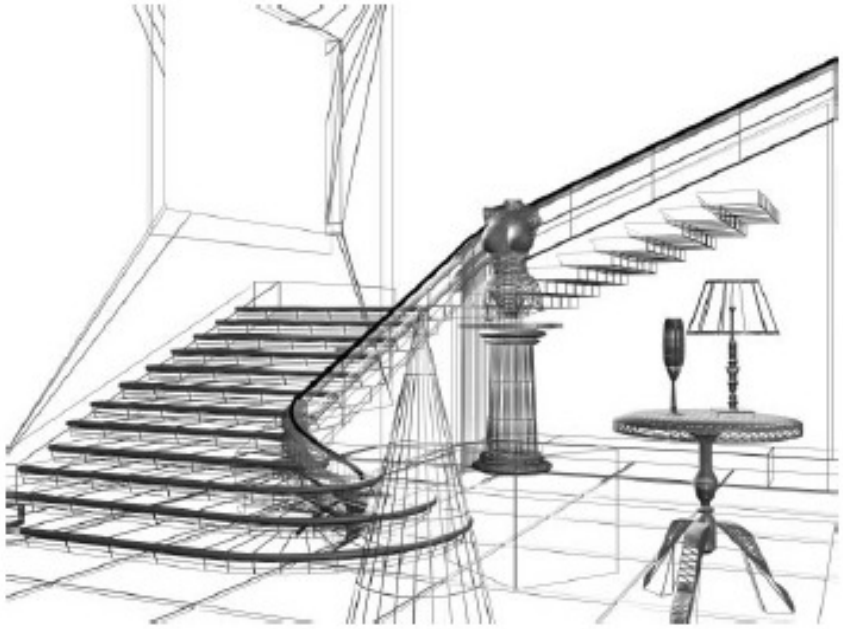




## *Wire frame* : adequado para posicionamentos e desenho, mas não realístico

Todas as linhas são mostradas.

Passo seguinte do realismo eliminar **partes da cena que não são vistas quando objetos opacos são vistos de determinada direção.**



# Tratamento de *hiddens* ou *Hidden Line/surface problem*

Eliminação de linhas:
caso particular da

definição de que faces ou superfícies são ocultas por outras do objeto ou cena.

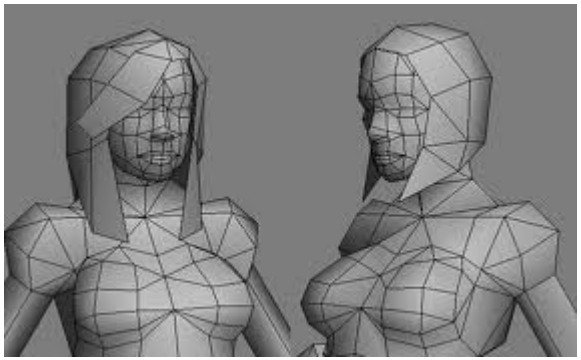
# Técnicas de visibilidade

*Back face culling*

*Priority fill* ou *painter's algorithm*

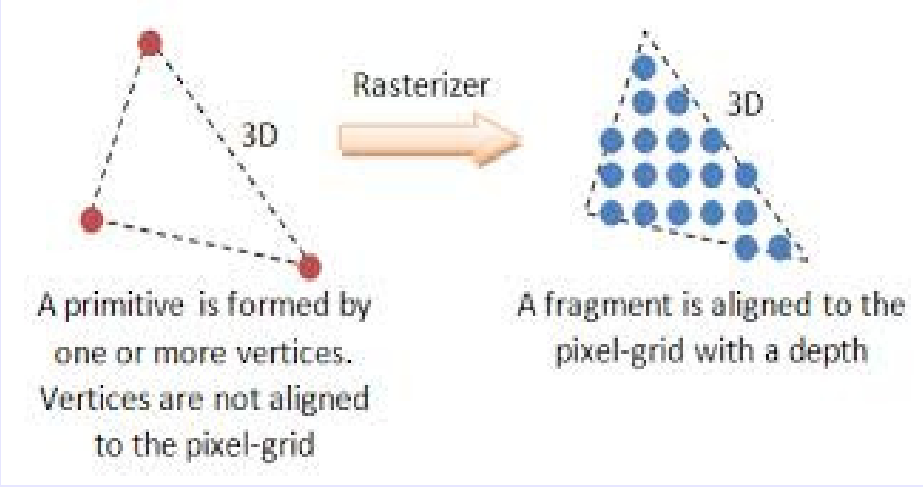
*Z- buffer*
*(mim Max)*

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



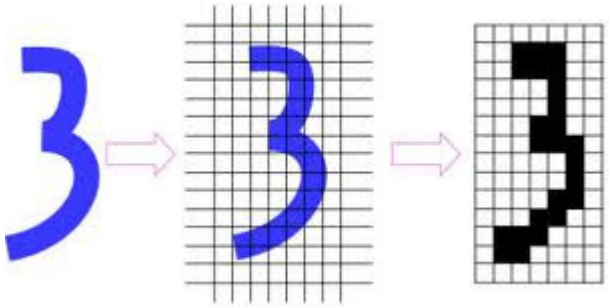
## HÁ ALGORITMOS NA FORMA **VETORIAL** E **RASTER**

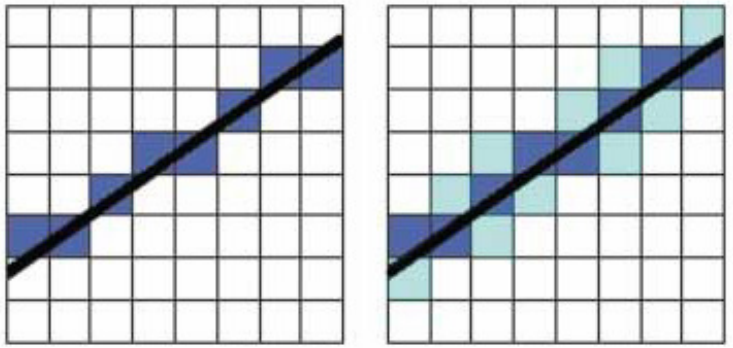
**RASTER:** o objeto em 3D é tratado na forma final quando já "*discretizado*" em pixels.



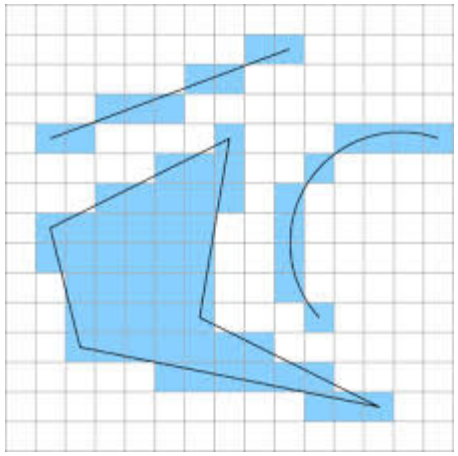

**Rasterisation**
(ou **rasterization**)
converte uma imagem descrita como vector format para a forma de pixels ( dots ) para representação em video, printer ou storage in a bitmap file format.

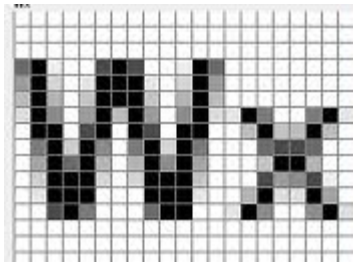
## *Alising → antialising*





*Rasterizar* = Usar a malha de pixels para descrever os objetos!





# *Back face culling*


Demo: em javascript:

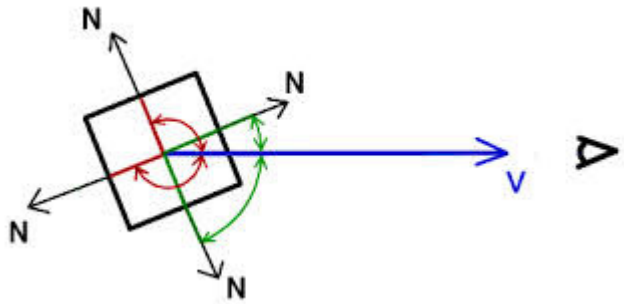
http://echolot-1.github.io/back-face-culling-demo/

echolot-1/**back-face-culling-demo**

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

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

# *Back face culling*



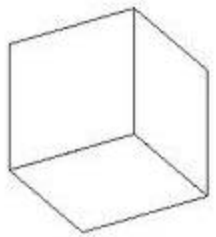

Idéia básica:

**Remover faces traseiras dos objetos em relação ao observador**

Adequadas para objetos convexos.

OBS :

Ser **não convexo** ≠ ser **côncavo**

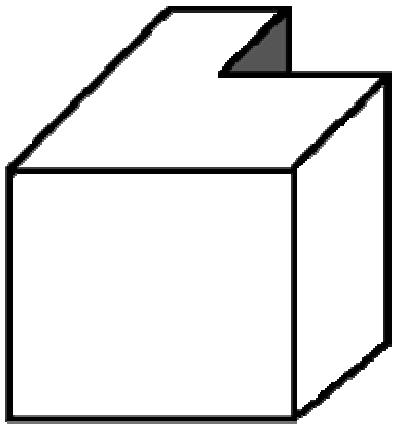


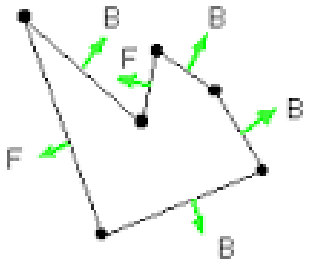
# Objetos convexos

**Definição:**

Formado por faces convexas.

*i.e.* Formado por polígonos convexos: nos quais a **ligação entre quais quer 2 pontos** internos nunca passa por uma parte externo a face:



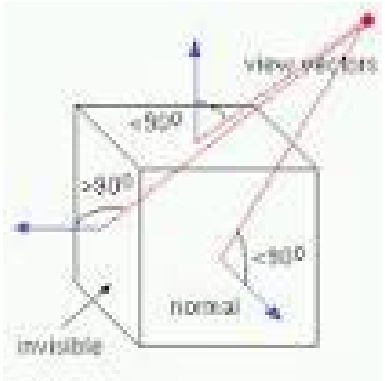


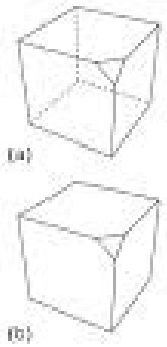
# Algoritmo no espaço do objeto

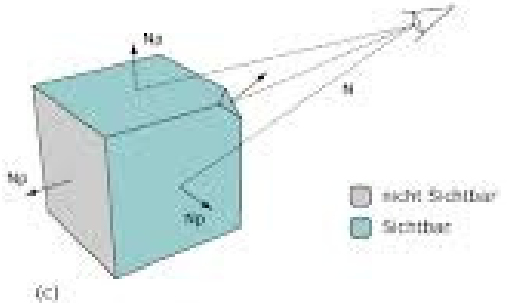
Usa-se a **direção que as normais** às faces fazem com a direção de visualização.

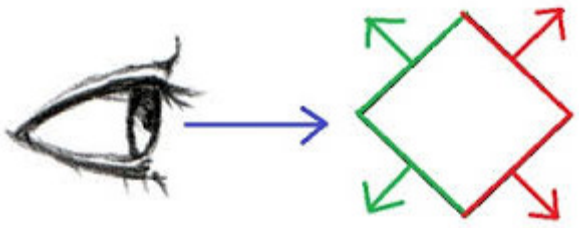
Entre **-90** graus e **90** graus a **face é visível** pelo observador (ou a face é de frente) .

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



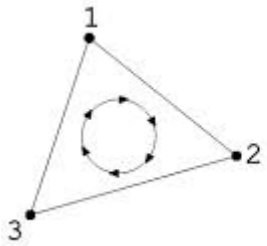



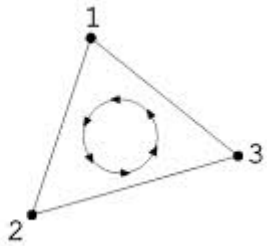



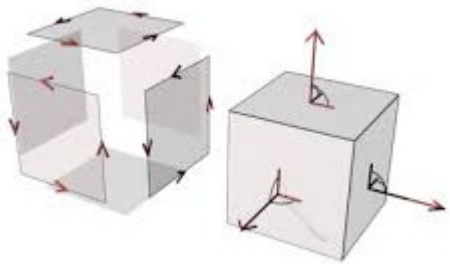


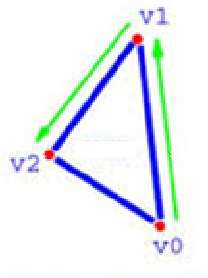
# 1-Obtêm a normal às faces

Através do cálculo do **produto vetorial** de dois vetores da face: a ordem dos vértices é importante!

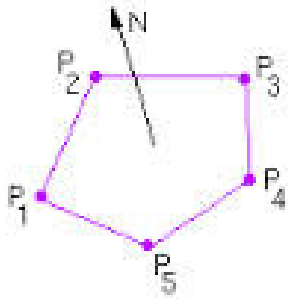





$$N = (V_1 - V_0) \times (V_2 - V_0)$$

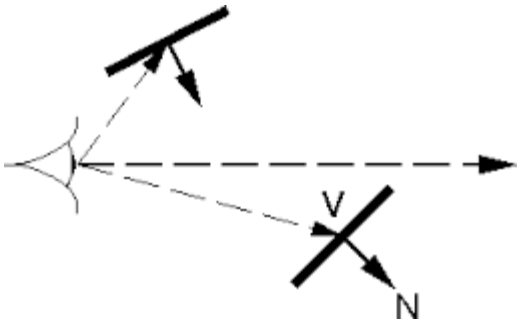


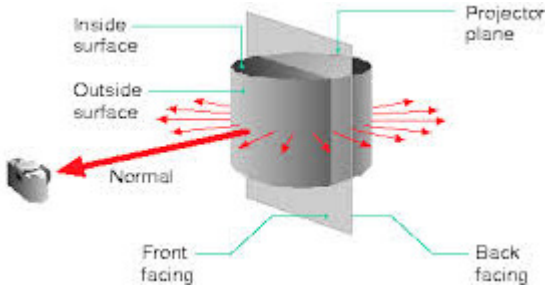

$$(V_1 - V_0) \times (V_2 - V_0) = -(V_2 - V_0) \times (V_1 - V_0)$$

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

## 3- Verifica-se o ângulo!

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 ° !

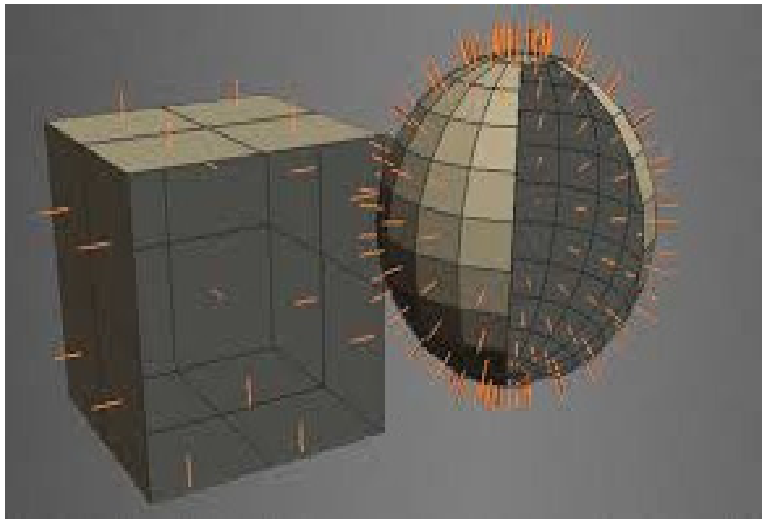


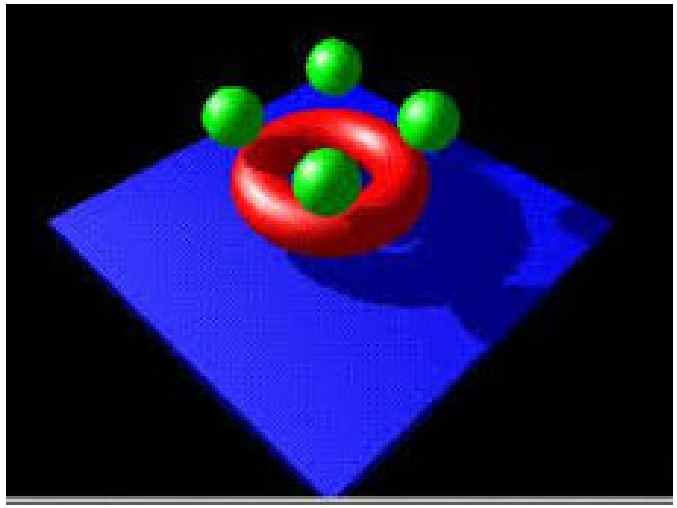




$$(V_0 - P) \cdot N \geq 0$$

# Revisitando a fórmula de Euler

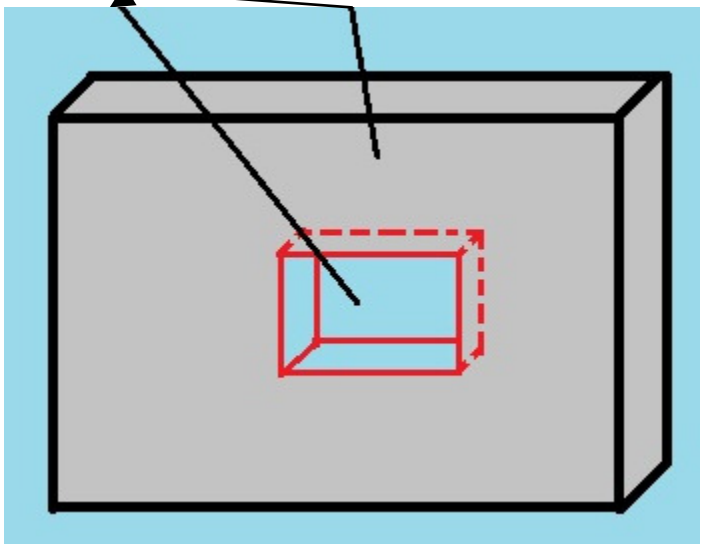
Objetos ***topologicamente equivalentes*** → se feitos de materiais deformáveis poderiam ser transformados uns nos outros.





# Revisitando a fórmula de Euler

***Genus*** G de um objeto : menor **número de furos** que trespassam o objeto.

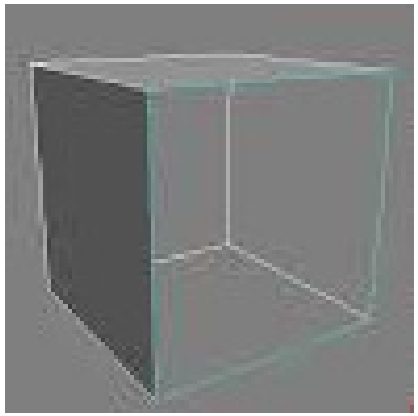


***Genus*** G=**1**

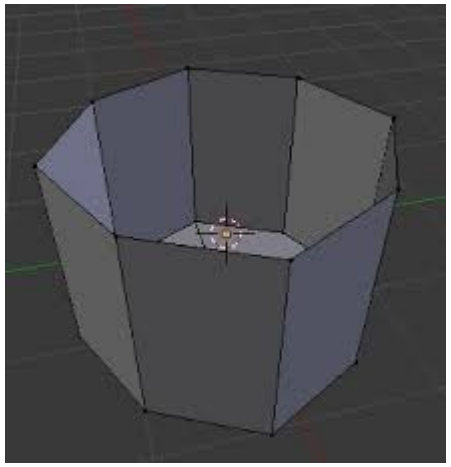
***Qual o genus*** de uma tubulação em Y?
Resposta: Veja o vídeo no Breno onde ele mostra isso por deformação!
Segue o link do vídeo no youtube: http://youtu.be/QkcryL4f6hE

Se há partes abertas – normais nas direções adequadas (até 2 normais) por face

***Buracos H*** : menor **número de furos** que não trespassam ou loops fechados de faces.



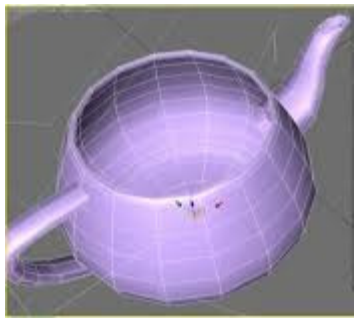
***Buracos*** H=**1**



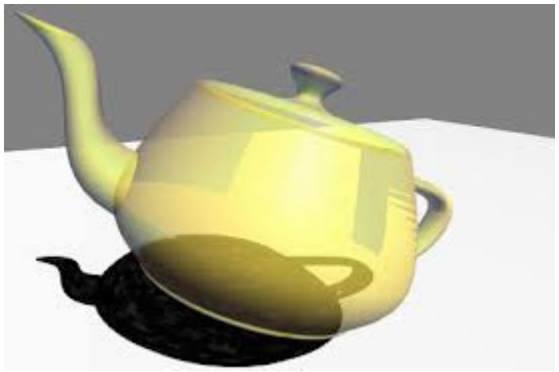
# Revisitando a formula de Euler→ **Euler - Poincaré**:

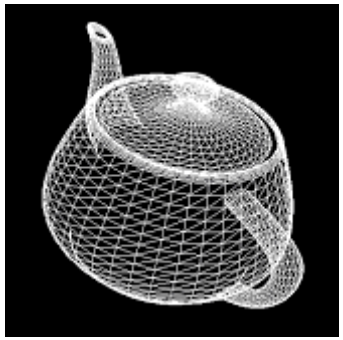
**Componentes separáveis** ou partes conectadas: C

formula de Euler - Poincaré: V-A+F-H = 2(C-G)



H=1 e G=?





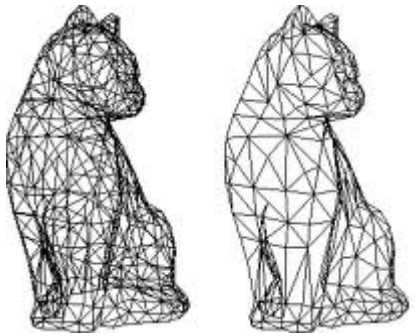
***Utah teapot***

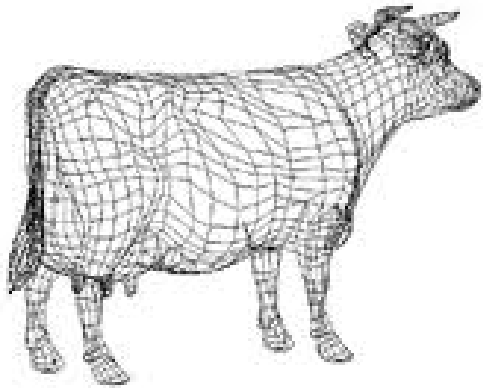
fórmula de Euler : V – A + F = 2

Um **teapot** não é uma **chaleira** ! Nunca é usado para por água no fogo e a ferver!

# Importante da modelagem correta para o de uso do objeto adequadamente

Já definir se há **buracos H**, ou furos **trespassantes G** ou **partes conectadas C**, na modelagem inicial do objeto.





Qual o ***Geno*** de um corpo humano para uma modelagem que o tratasse por dentro, como para uma endoscopia?

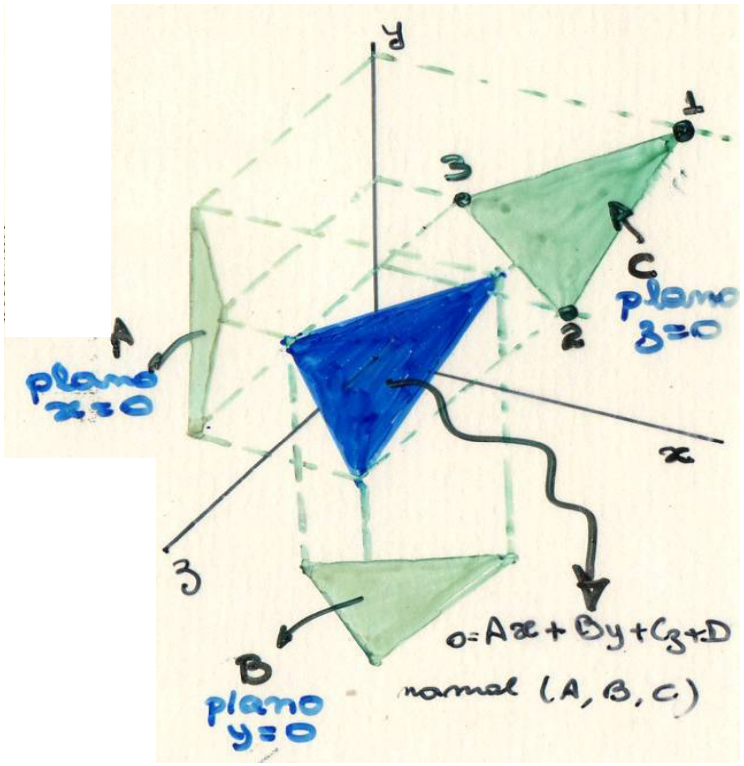
# Relações entre a normal e a quação de um plano

Plano no espaço tem suas normais

Nas direções x,y,z

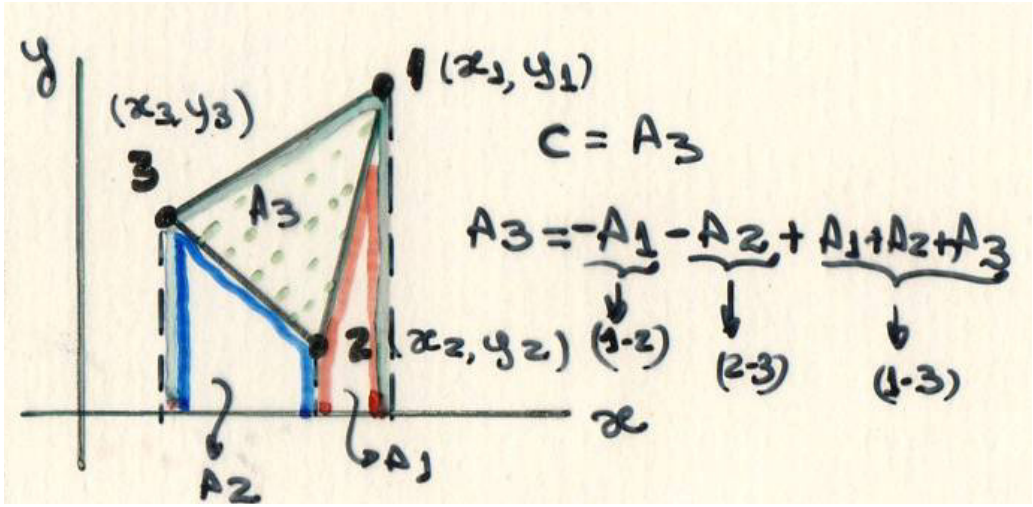
Proporcionais as areas

De suas projeções nestes plano



# Cada uma das areas projetadas

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




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

$n+1 = 1$

# Passando para os dados já projetados

ALGORITMOS NA FORMA **RASTER ou**

**semi raster**

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

**Isto é esses dependem de voce já ter passado de 3D para 2D.**

**Da direção de vista da cena!!**

# *Painter's algorithm*

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

Na projeção de cena 3D para o plano do video 2D é nescessario **decidir que faces são visiveis ou escondidas (** hidden ) .

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

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

Cobrindo assim as parte invisiveis — ou seja o visibility problem  é resolvido com algum custo extra (the cost of having painted invisible areas).

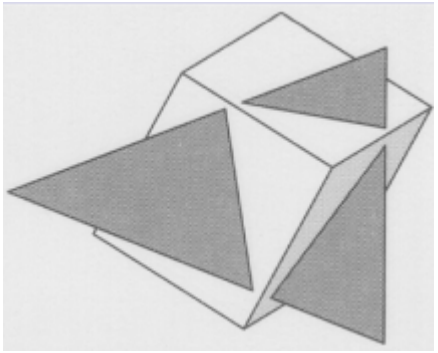
A ordem usada é chamada ***depth order. Essa ordenação tem uma boa propriedade***: if one object obscures **part** of another **then it is painted after** the object that it obscures. **RESOLVE ALGUNS CASOS DE PARCIAIS**

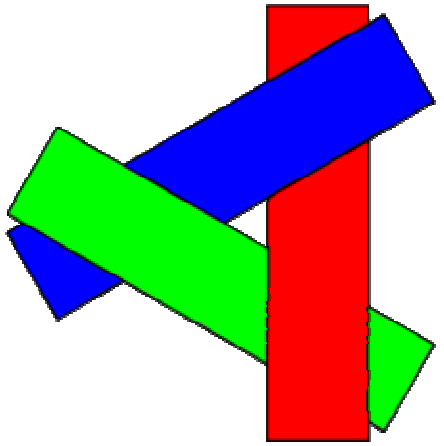
# *Painter's algorithm*

*Possibilidade de falha → quando parte MAIORES de uma face se sobrepoem a outra→ solução divisão da face* (**Newell's Algorithm**).

Essa falha do algoritmo levou ao desenvolvimento do método de

**z-buffer** ou **depth buffer**





# z-buffer algorithm

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

Considera um array : z 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 RGBα

Considere 2 polígonos, um **vermelho=R (red, 1** , 0 , 0, **0.5 ),** e o outro **azul=B(blue,** 0 , 0 , **1**, **0.5 )** *renderizaveis* 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** (Rederizando de traz para a frente as percentagens da cada cor):

**Green background.** (0 , 1 , 0 )

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

**Poligono 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 {P1,P2,.....Pn} 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 α:
      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 α:
        }
    }
display COLOR array.
```

# *Masking Technique*
# *ou mim Max*

Muito bom para o Hidden lines de curvas.

Isso é eliminar linhas invisiveis de superficies

É um ALGORITMO NA FORMA **RASTER**

**depende**

**Da direção de vista da cena!!**

**de voce já ter passado de 3D para 2D.**

# Imagine que foi gerada uma superficie

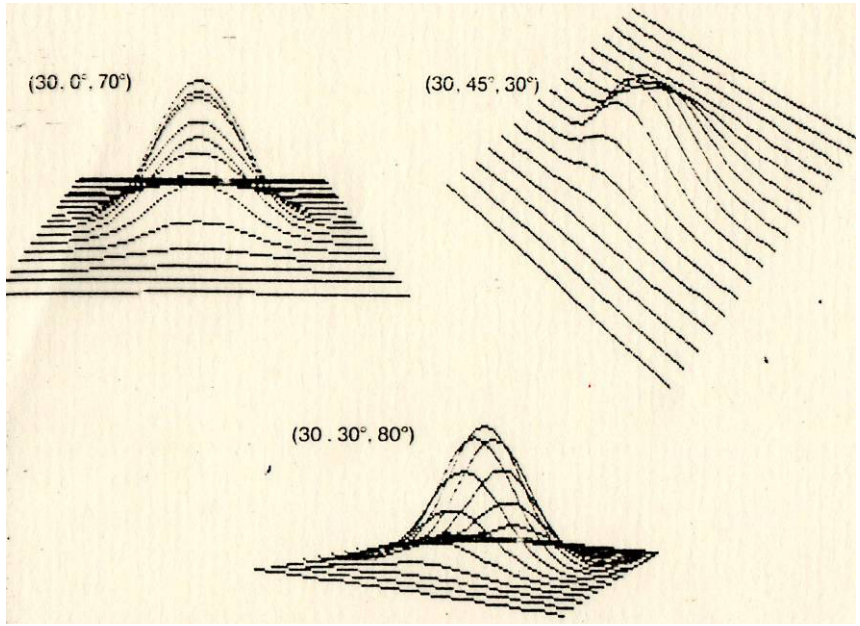
A partir de uma série de curvas.

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

Ou sua projeção de

Determinada direção

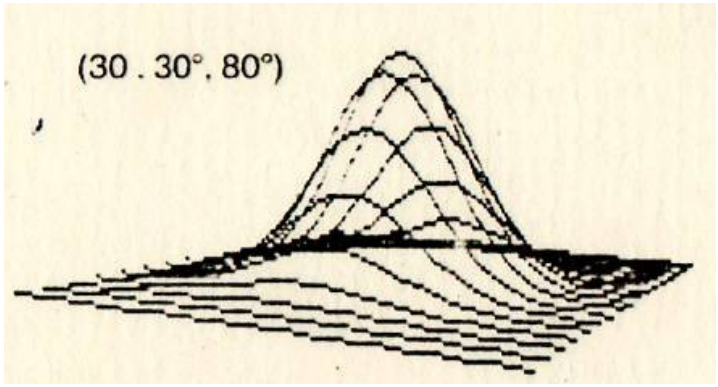
Ou seja ela já é descrita por uma série de linhas em 2D de determinada direção.

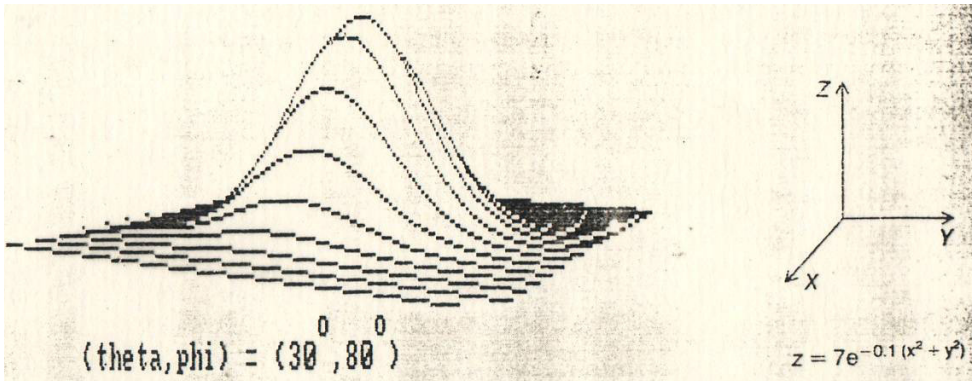



Mesma superficie representada por um conjunto de curvas e Vista de diversos pontos de vista

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

Hidden lines por mascaramentoou lista de limites verticais superioires e inferiores de cada passo (pixel)horizontal

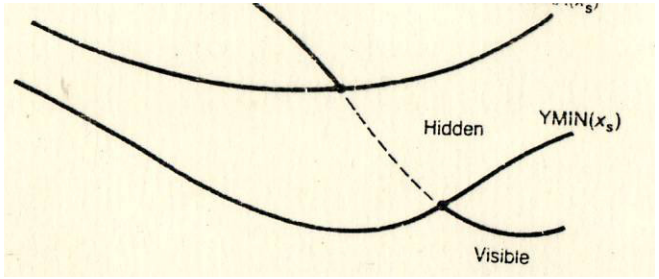


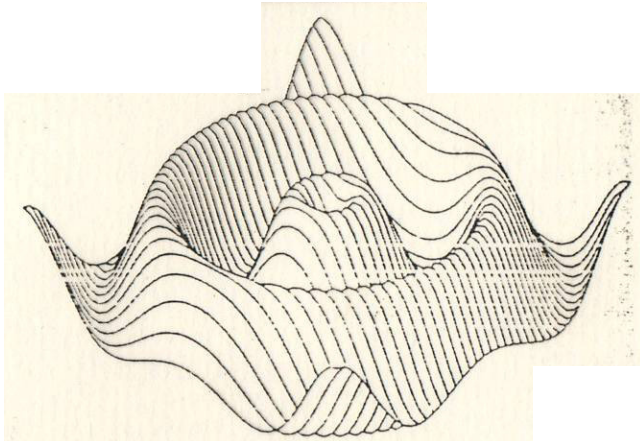
# O conceito da tecnica de mascaramento

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

$Y_{MAX}(xp)$




$Y_{MIN}(xp)$



# O numero de pixel usado, ou o passo

Pode ser uma função da curvatura da superficie ou curva.

Mais curvatura menor passo!!

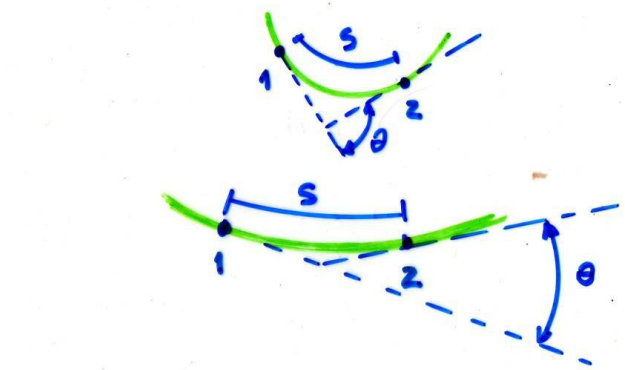
E como se obtem a curvatura?

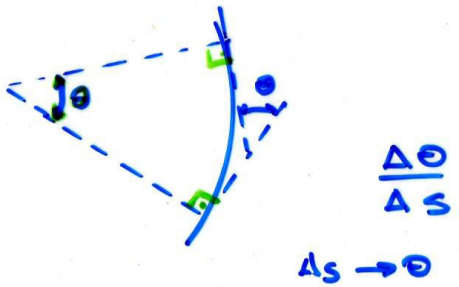
# O que é curvatura?

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

# Ray tracing *simplificado ou aproximado ou*

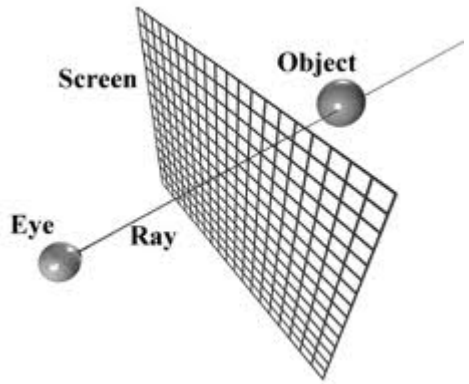
**Ray casting** lança raios a partir do observador de forma a perceber a distância dos objetos que compõem a cena.

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

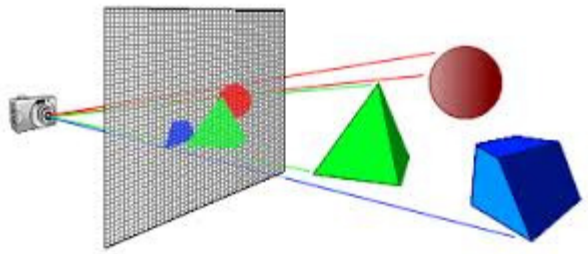
Supõe-se um raio do olho do observador passando por **cada ponto da tela** a ser desenhada. O ponto da tela receberá a cor do objeto que for atingido na cena pelo raio.

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.

# 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/