O Pipeline de Renderização

Computação Gráfica

Marcelo Walter – UFPE

The Graphics Pipeline

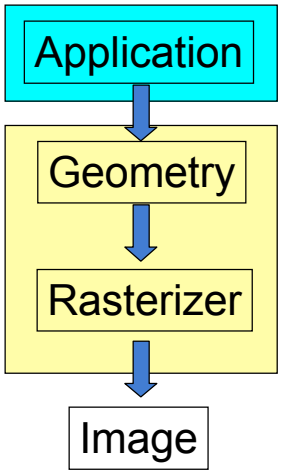
- Processo de sintetizar imagens bidimensionais a partir de câmeras e objetos virtuais
- Visão em alto nível inicial para aprofundarmos nas próximas aulas

The Graphics Pipeline

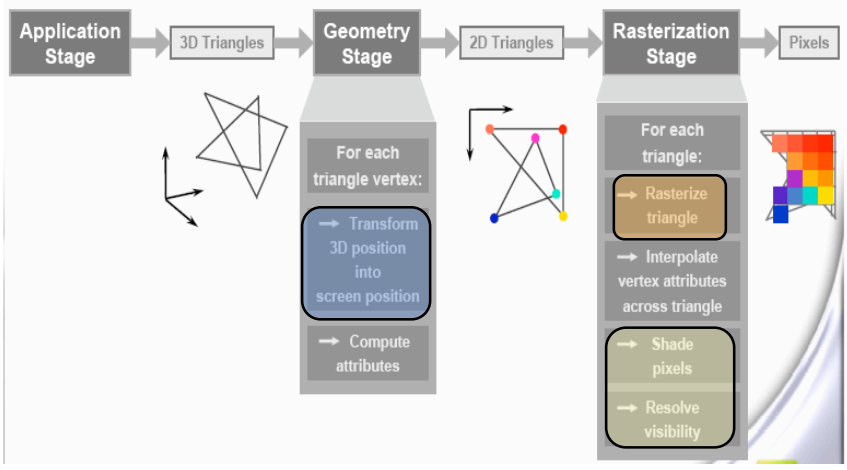
- Três estágios conceituais
- O desempenho é determinado pelo estágio mais lento
- Sistemas gráficos modernos:

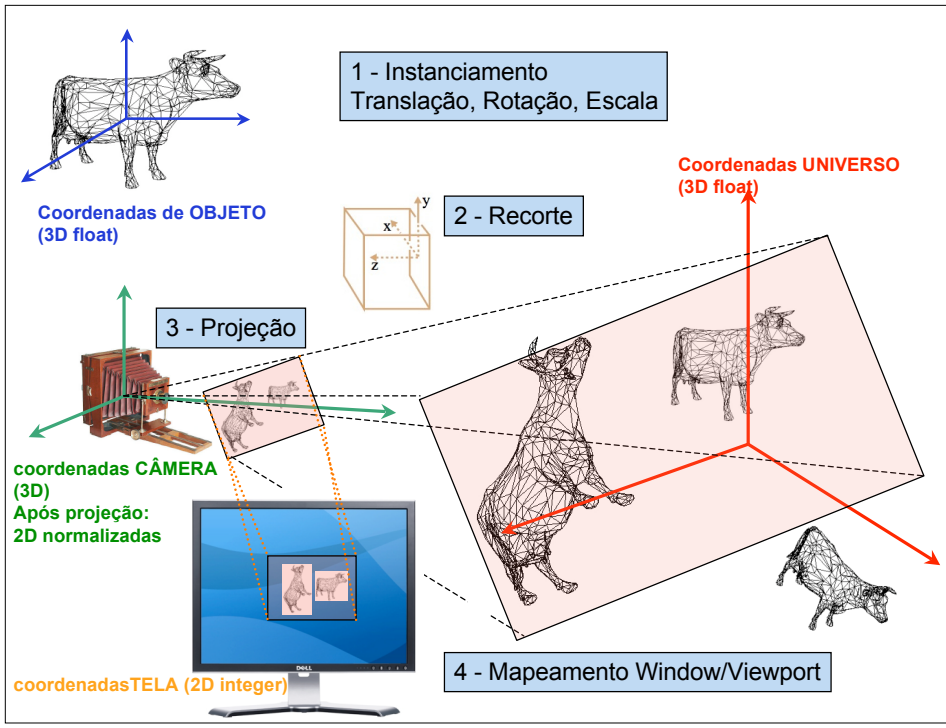
  software:

  hardware:



The Graphics Pipeline





Resumo

- Transformações Modelagem
- Iluminação (*Shading*)
- Transformação Câmera
- Recorte
- Map. Tela
- Rasterização
- Visibilidade

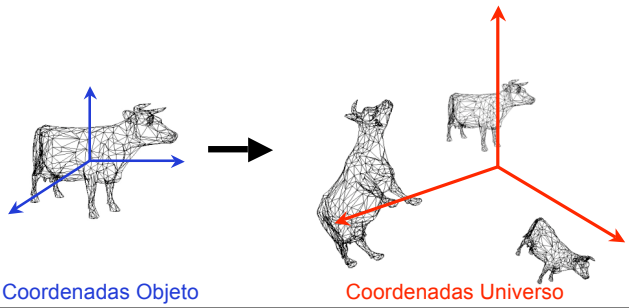
Adaptação e melhoramentos de uma aula sobre o mesmo assunto (MIT - EECS 6.837 Durand and Cutler)

Resumo

- Transformações Modelagem
- Iluminação (*Shading*)
- Transformação Câmera
- Recorte
- Map. Tela
- Rasterização
- Visibilidade

✓Objetos definidos no seu próprio sistema de coordenadas

✓Transformações de modelagem orientam os modelos geométricos num sistema comum de coordenadas (UNIVERSO)



Coordenadas Objeto — Coordenadas Universo

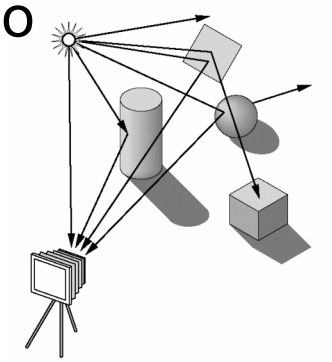
Resumo

- Transformações Modelagem
- Iluminação (*Shading*)
- Transformação Câmera
- Recorte
- Map. Tela
- Rasterização
- Visibilidade

✓Vértices iluminados de acordo com as propriedades geométricas e de material

✓Modelo de Iluminação Local (Flat, Gouraud)

Modelo de Iluminação



Objetivo principal

Cálculo de iluminação nos vértices baseando-se na posição, orientação e características das superfícies e fontes de luz que as iluminam

Posteriormente Modelos de Sombreamento

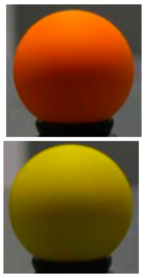
Cálculo de iluminação nos demais pixels que compõe o triângulo a partir das cores nos vértices

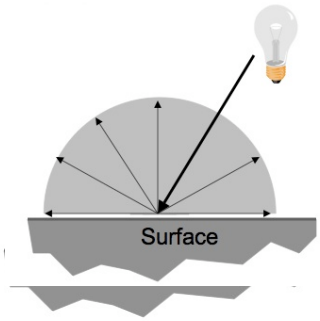
Considerações Iniciais

- Um comprimento de onda apenas (monocromático)
- Fontes de luz pontuais (uma direção basicamente)
- Uma fonte de luz apenas

Reflexão Difusa Ideal

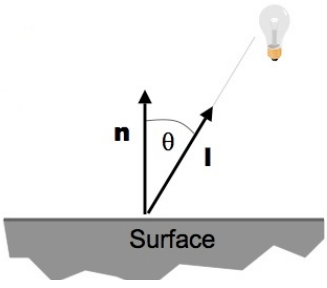
- Luz refletida igualmente em todas as direções
- Exemplos: giz, quadro-negro, algumas tintas

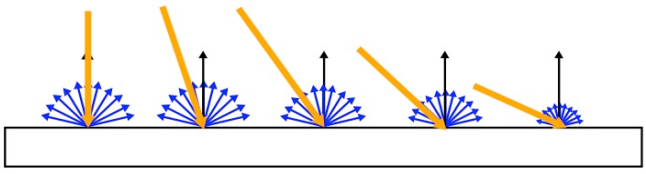




Lei dos Cossenos de Lambert

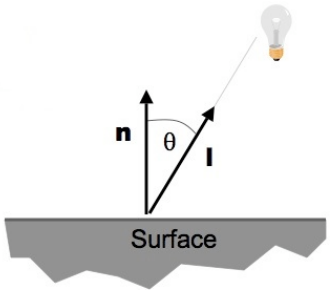
- Intensidade da reflexão difusa é proporcional ao cosseno do ângulo entre a fonte de luz e a norr superfície



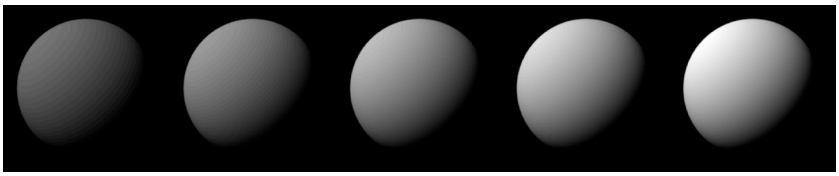


Computacionalmente...

- $I_d = I_p \cdot k_d \cdot \cos\theta$
- $\cos\theta = (n \cdot l)$
  Produto escalar dos 2 vetores NORMALIZADOS
- I_p = intensidade da fonte de luz
- k_d = coeficiente de quão difuso é o objeto [0,1]



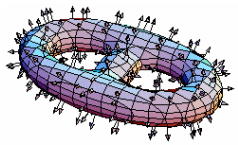
Reflexão Difusa – Exemplos



Kd= 0.4 Kd= 0.55 Kd= 0.7 Kd= 0.85 Kd= 1

menos difuso mais difuso

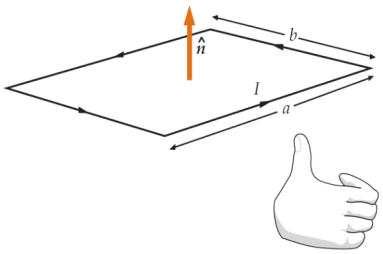
Vetor Normal



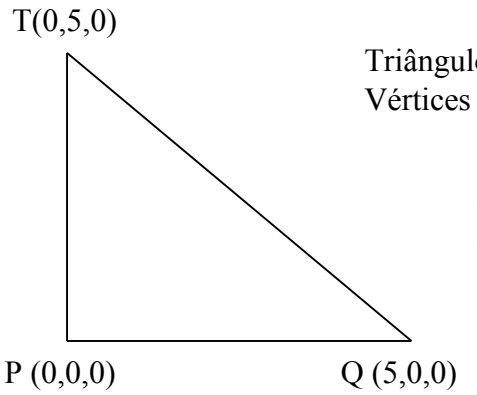
- Fundamental nos cálculos de iluminação
- Em OpenGL

```
glNormal3f( nx, ny, nz);
```

- Consistência na especificação (apontando "para fora")



Exemplo



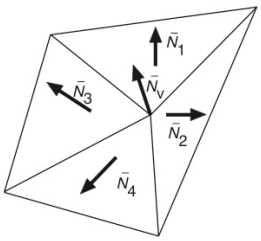
Triângulo A
Vértices PQT

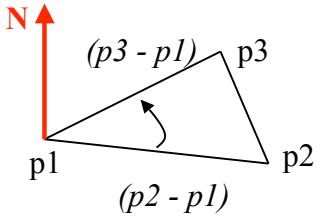
Triângulo B
Vértices PTQ

Qual enumeração é correta?

Aproximando o Vetor Normal

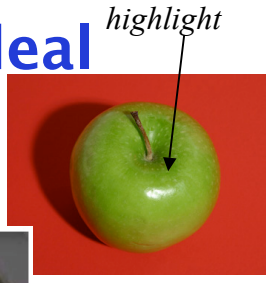
- Cálcula-se para as faces e após, média das faces encontra a normal do vértice
- Para uma face: produto vetorial de vetores a partir das arestas

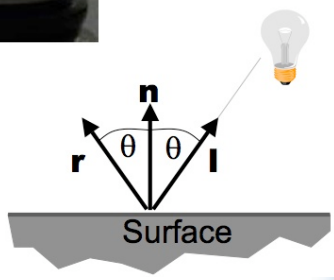




Reflexão Especular Ideal

- Efeito visual que devolve a energia luminosa numa direção preferencial
- Depende da posição do observador
- ângulo de incidência = ângulo de reflexão





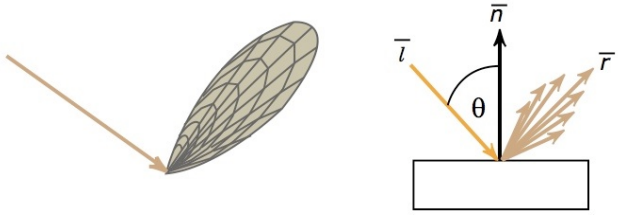
Qual a cor do reflexo especular??

Reflexão Especular não-ideal

- Reflexão apresenta distribuição ao redor da direção ideal

Reflexão Especular não-ideal

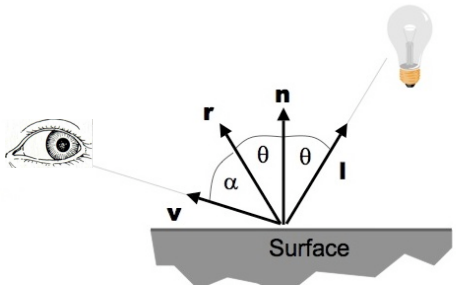
- Modelo empírico simples proposto por *Phong* [1975]*
- A reflexão especular varia pouco ao redor da direção especular pura



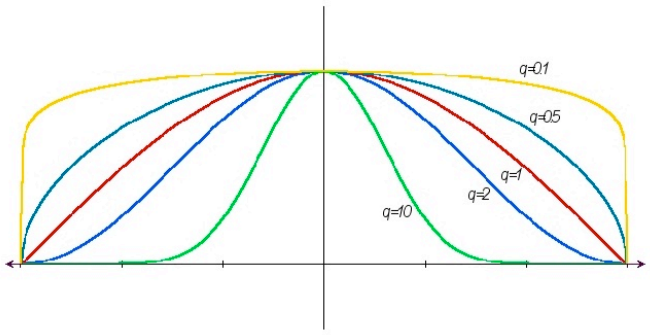
*Phong tem duas contribuições importantes em CG. Esta é a primeira. A outra está relacionada com Modelo de Iluminação para polígonos, veremos adiante.

Computacionalmente…

- $I_s = I_p \cdot k_s \cdot \cos \alpha^q$
- $\cos \alpha = (r \cdot v)$
  Produto escalar dos 2 vetores NORMALIZAD
- I_p = intensidade da fonte de luz
- k_s = coeficiente de quão *especular* é o objeto [0,∞…], na prática algumas centenas



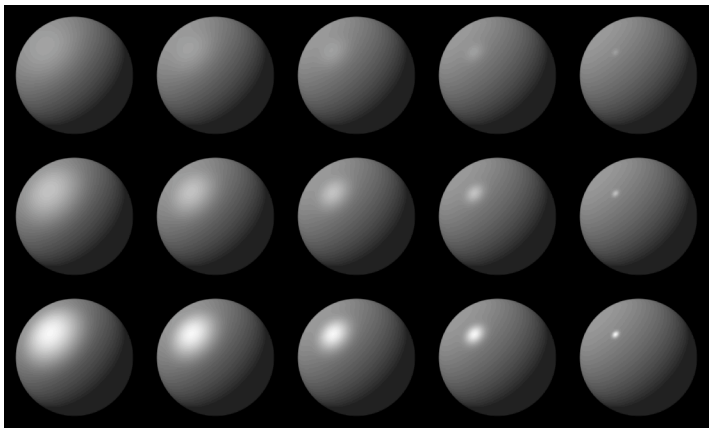
Expoente especularidade Phong



$\cos \alpha^q$

Reflexão Especular

Kd=0.45



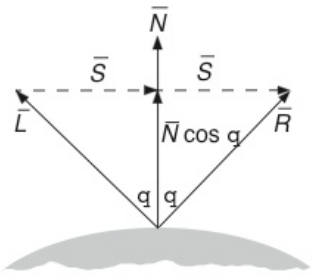
Ks=0.1

Ks=0.25

Ks=0.5

q=3 q=5 q=10 q=27 q=200

Como calcular R?



Todos vetores unitários

R = N cos q + S
S = N cos q - L

R = 2 N cos q - L

cos q = (N . L)

R = 2N (N.L) - L

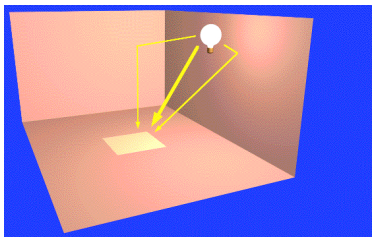
Múltiplas Fontes de Luz

- Existindo *m* fontes de luz, basta somarmos os termos de cada fonte de luz
- Qual um problema em potencial desta abordagem??

Qual a solução de OpenGL?
"*…the color values are clamped to the range [0,1].*" (Red book, p. 205)

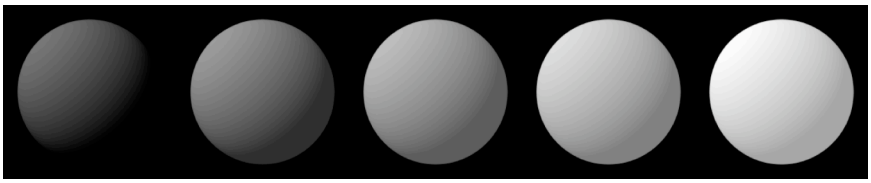
Luz Ambiente

- Fonte de luz sem direção específica. Aproxima (muito mal…) as múltiplas reflexões entre as superfícies presentes no ambiente
- Conhecida como luz ambiente
  - $I = I_a.K_a$
    - I_a – intensidade da luz ambiente
    - K_a – coeficiente de reflexão ambiente



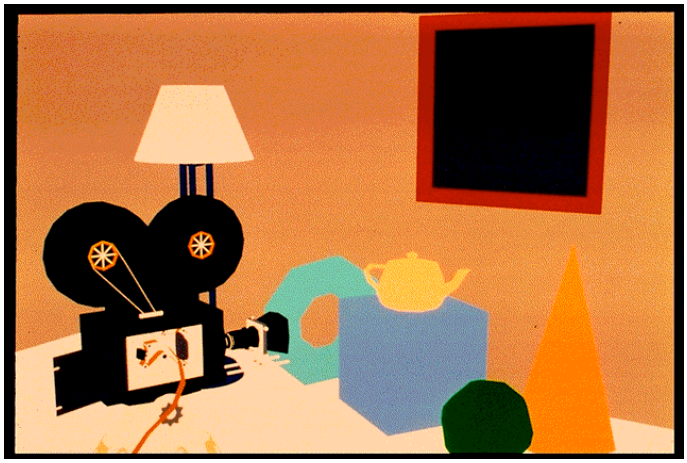
Componente Ambiente

Ia=Ip=1.0, Kd=0.4



Ka=0.0 Ka=0.015 Ka=0.3 Ka=0.45 Ka=0.6

Luz Ambiente



Aqui não tem componente difusa…

Colocando tudo junto...

Considerando vetores normalizados e **m** fontes de luz:

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

Colocando tudo junto...

agora com cores

$$I_\lambda = I_{a\lambda} k_{a\lambda} + \Sigma \{ I_{m\lambda} [k_{d\lambda} (N.L) + k_{s\lambda} (R.V)^q] \}$$

λ = (R,G,B)

Transformações Modelagem

Iluminação (*Shading*)

Transformação Câmera

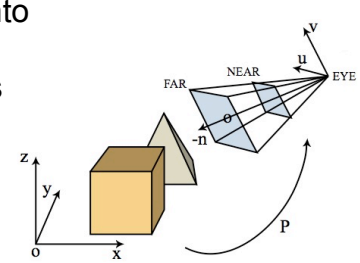
Recorte

Map. Tela

Rasterização

Visibilidade

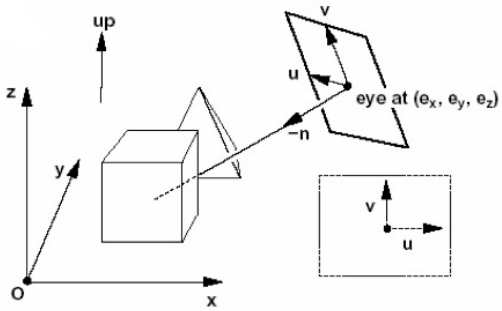
✓Mapeamento de coordenadas de Universo para câmera



✓Escolha da projeção: perspectiva ou ortográfica

Sist. Coordenadas Câmera (SCC)

- Envolve uma translação à origem do sistema e uma mudança de base ortogonal



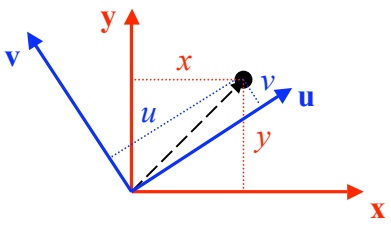
Definindo o Sistema de Coordenadas de Câmera – SCC

- Precisamos especificar:
  - Origem do SCC: ponto qualquer no espaço
  - Direção para onde a câmera está apontando
  - Direção "para cima": *up vector*
- A partir destas três informações um sistema de coordenadas ortogonal pode ser estabelecido
- Ver por exemplo: *Foley at al. Introduction to Computer Graphics. p. 201*

Definindo SCC

- Transformar as coordenadas, ou seja, dado os sistemas de coordenadas xyz e uvn e um ponto p=(x,y,z) encontrar p=(u,v,n)

Em 2D esquematicamente:



Definindo SCC

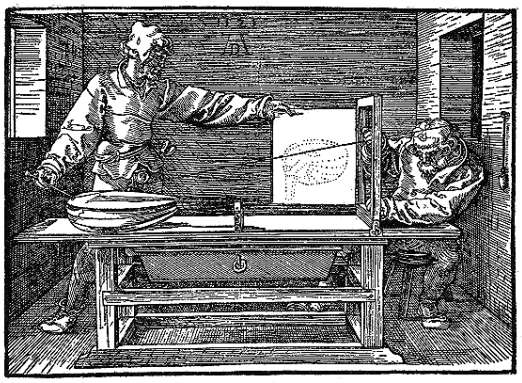
$$\begin{pmatrix} u \\ v \\ n \end{pmatrix} = \begin{pmatrix} u_x & u_y & u_z \\ v_x & v_y & v_z \\ n_x & n_y & n_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- Onde u_x = **x . u**, u_y = **y . u**, u_z = **z . u** etc

Produto escalar

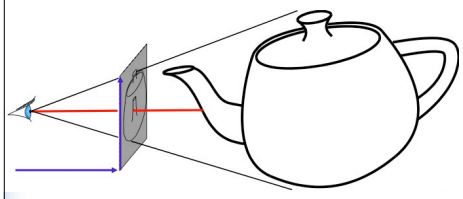
Volume de Visualização

- Finalmente, precisamos definir uma área do espaço que será considerada
- Somente os objetos dentro do volume serão considerados
- Implica definirmos um tipo de **PROJEÇÃO**



Albrecht Durer doing perspective projections in 1525.
http://www.viewmuseum.com/

Projeções



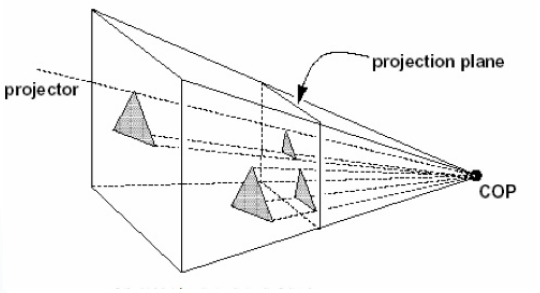
- Genericamente, projeções transformam pontos em um sistema de coordenadas com N dimensões em pontos num sistema com dimensão menor do que N
- Para nós interessa apenas o caso de 3D -> 2D

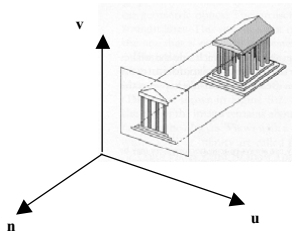
Projeções

- Uso de linhas de projeção, denominadas *raios de projeção*
- Origem em um *centro de projeção* (COP) e passam por cada parte do objeto (no nosso caso vértices) interseccionando um *plano de projeção* onde finalmente tem-se as coordenadas 2D

Tipos de Projeções

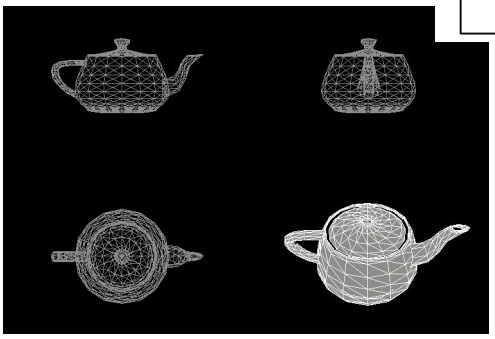
- Em CG 2 tipos: *perspectiva* e *ortográfica*
- Diferença: na projeção ortográfica o COP está no infinito, logo os raios de projeção são paralelos uns aos outros

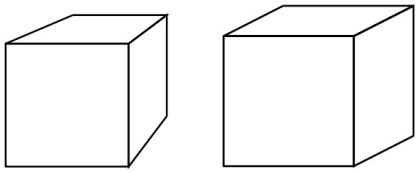




Tipos de Projeções

- Diferença Visual





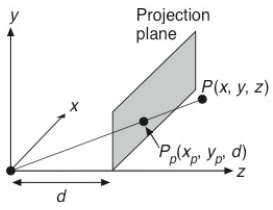
Qual é qual?

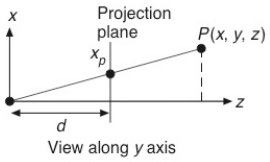
Obtendo Coordenadas de Projeção

- Precisamos "livrar-nos" de uma dimensão
- Para projeções *ortográficas* é simples:
  - Basta descartarmos a coordenada equivalente a ***n*** no SCC (assumindo o plano de projeção em n = 0)

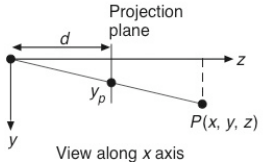
$$\begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Obtendo Coordenadas de Projeção





View along *y* axis

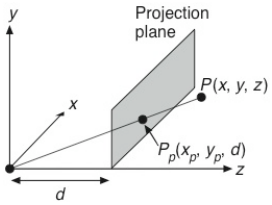


View along *x* axis

- Para projeções perspectivas?
- Assumindo que o plano de projeção encontra-se perpendicular ao eixo ***n*** do SCC e a uma distância ***d***
- Por semelhança de triângulos temos:

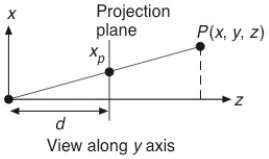
$$\frac{x_p}{d} = \frac{x}{z} \qquad \frac{y_p}{d} = \frac{y}{z}$$

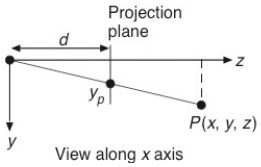
Obtendo Coordenadas de Projeção



$$x_p = \frac{d \cdot x}{z} = \frac{x}{z/d} \qquad y_p = \frac{d \cdot y}{z} = \frac{y}{z/d}$$

matricialmente:





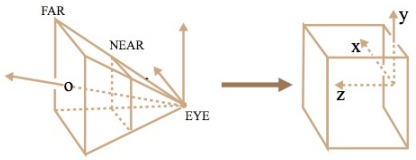
homogenize

$$\begin{pmatrix} x*d/z \\ y*d/z \\ d \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z/d \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Resumo

- Transformações Modelagem
- Iluminação (*Shading*)
- Transformação Câmera
- **Recorte**
- Map. Tela
- Rasterização
- Visibilidade

✓Transformação para Coordenadas Normalizadas



câmera

Normalized Device Coordinates

Transformações Modelagem

Iluminação (*Shading*)

Transformação Câmera

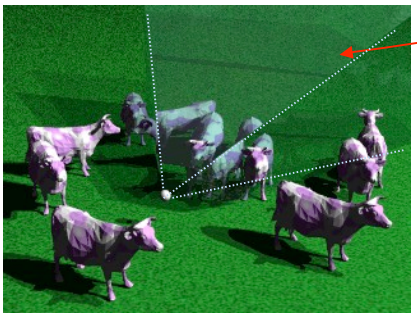
Recorte

Map. Tela

Rasterização

Visibilidade

✓Eliminação de objetos inteiros e partes de objeto que estão fora do *Frustum*

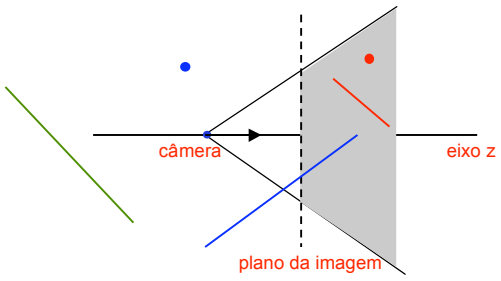


Universo

Imagem final

Estratégias para Recorte

- Recortar durante a rasterização (livro do Foley chama de *scissoring*, recorte em 2D)
- Recorte analítico: altera a geometria 3D de entrada



Importância do Recorte

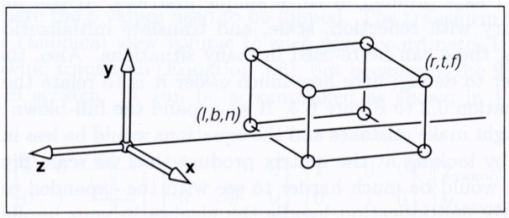
- Etapa de otimização
- Preprocessamento para determinação de visibilidade
  - Não exibe primitivas 'atrás' da câmera
- Garante que somente primitivas potencialmente visíveis serão rasterizadas

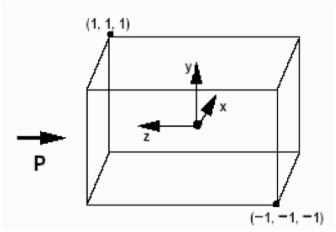
Coordenadas Normalizadas

- Vantagens
  - Recorte mais eficiente para um volume retangular, alinhado com os eixos (*Volume-padrão*)
  - *Frustum* tem geometria arbitrária
- Desvantagens
  - Como todos os pontos são transformados antes do recorte, provavelmente estaremos transformando pontos que posteriormente serão eliminados pelo recorte

Coordenadas Normalizadas

Projeção Ortográfica





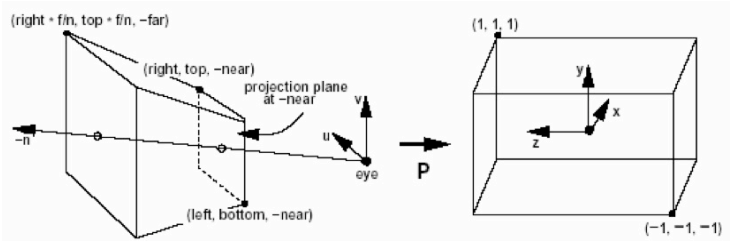
matriz de conversão*

*A obtenção desta matriz é um bom exercício de Álgebra :-)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & \frac{-(right+left)}{right-left} \\ 0 & \frac{2}{bottom-top} & 0 & \frac{-(bottom+top)}{bottom-top} \\ 0 & 0 & \frac{2}{far-near} & \frac{-(far+near)}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Coordenadas Normalizadas

Projeção Perspectiva



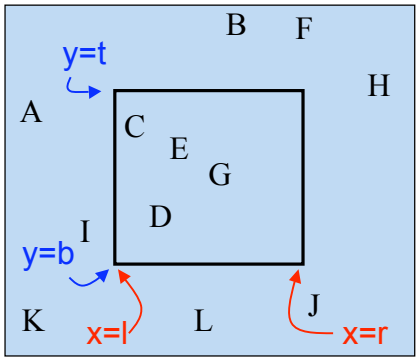
matriz de conversão

$$\begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} = \begin{bmatrix} \frac{2 \cdot near}{right-left} & 0 & \frac{-(right+left)}{right-left} & 0 \\ 0 & \frac{2 \cdot near}{bottom-top} & \frac{-(bottom+top)}{bottom-top} & 0 \\ 0 & 0 & \frac{far+near}{far-near} & \frac{-2 \cdot near \cdot far}{far-near} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

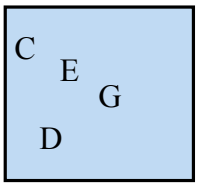
Algoritmos de recorte

(veremos em 2D primeiro e depois estendemos para 3D)

Recorte de pontos



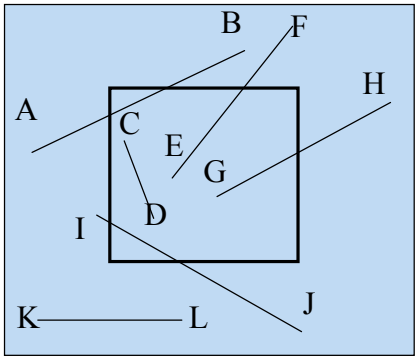
Qual a condição a ser satisfeita?

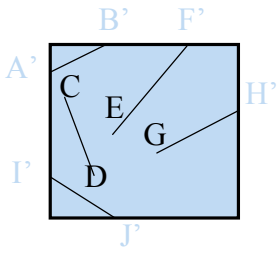


l <= x <= r
b<= y <= t

Algoritmos de recorte

Recorte de linhas

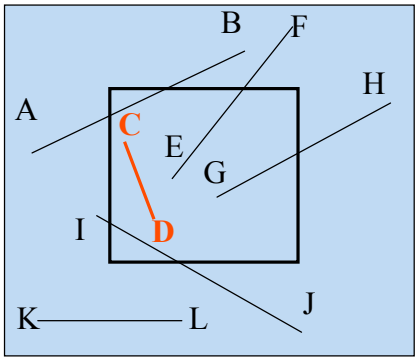




Saída desejada

Algoritmos de recorte

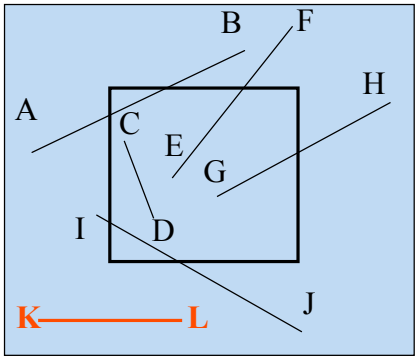
Recorte de linhas: *Trivialmente aceito*



Pontos estão dentro da janela

Algoritmos de recorte

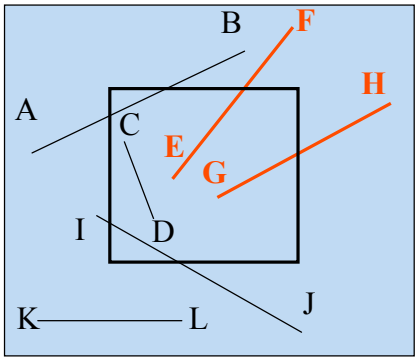
Recorte de linhas: *Trivialmente recusado*



Os dois pontos estão fora do retângulo e linha não cruza janela

Algoritmos de recorte

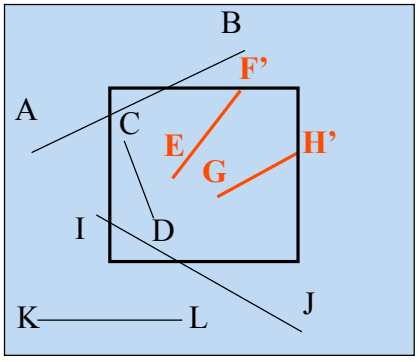
Recorte de linhas: *Cálculos de recorte*



Um dos pontos da linha está fora
e outro está dentro da janela

Algoritmos de recorte

Recorte de linhas: *Cálculos de recorte*

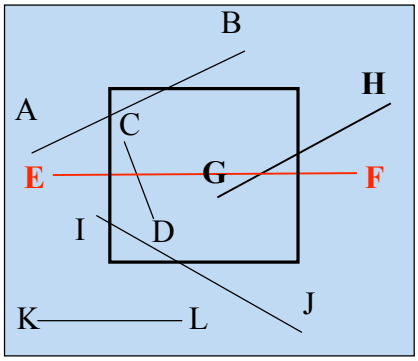


Um dos pontos da linha está fora
e outro está dentro da janela

Linha deve ser recortada

Algoritmos de recorte

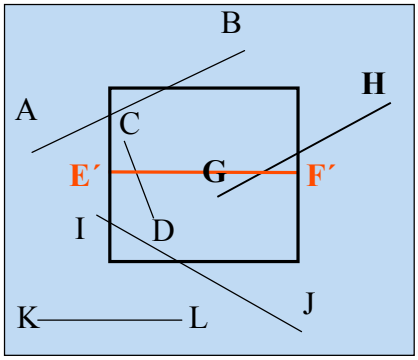
Recorte de linhas: *Cálculos de recorte*



Os dois pontos estão fora

Algoritmos de recorte

Recorte de linhas: *Cálculos de recorte*

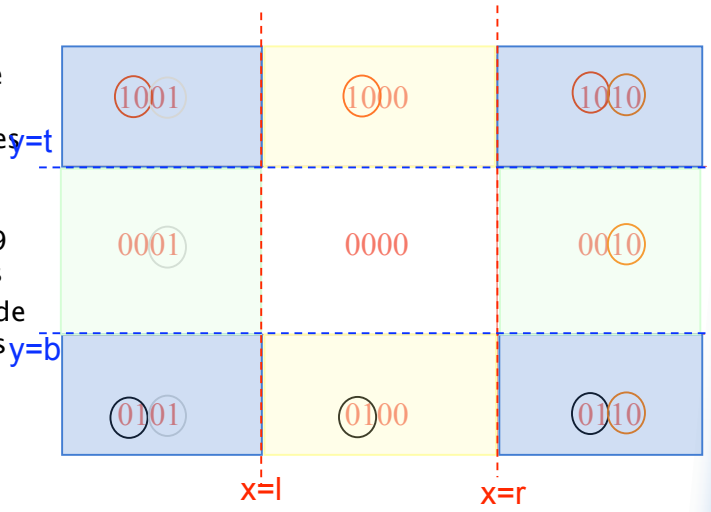


Os dois pontos estão fora

Linha deve ser recortada

Algoritmo de Cohen-Sutherland

- Maneira eficiente de determinar os diferentes casos
- Divide a região em 9 subespaços
- Atribuição de códigos aos espaços



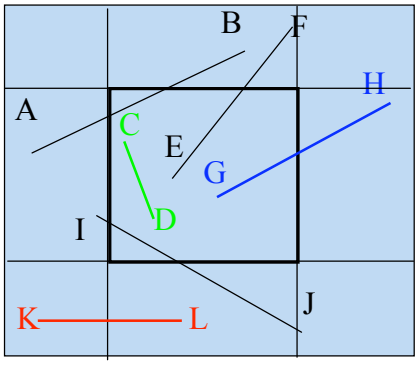
Cohen-Sutherland
Outcodes

If y > t ➔ seta primeiro bit em 1
If y < b ➔ seta segundo bit em 1
If x > r ➔ seta terceiro bit em 1
If x < l ➔ seta quarto bit em 1

http://www.cs.princeton.edu/~min/cs426/jar/clip.html

Applet exemplificando o algoritmo

Cohen-Sutherland
Usando os outcodes

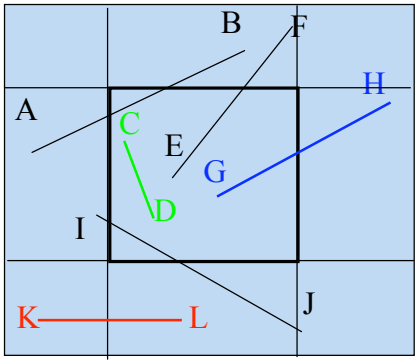


• Como devem ser os outcodes dos pontos trivialmente aceitos?

•Dos pontos trivialmente recusados?

• O que fazer com os que não são nem aceitos nem recusados?

Cohen-Sutherland
Usando os outcodes



• Como devem ser os outcodes dos pontos trivialmente aceitos?
0000

•Dos pontos trivialmente recusados?
AND bitwise diferente de 0!

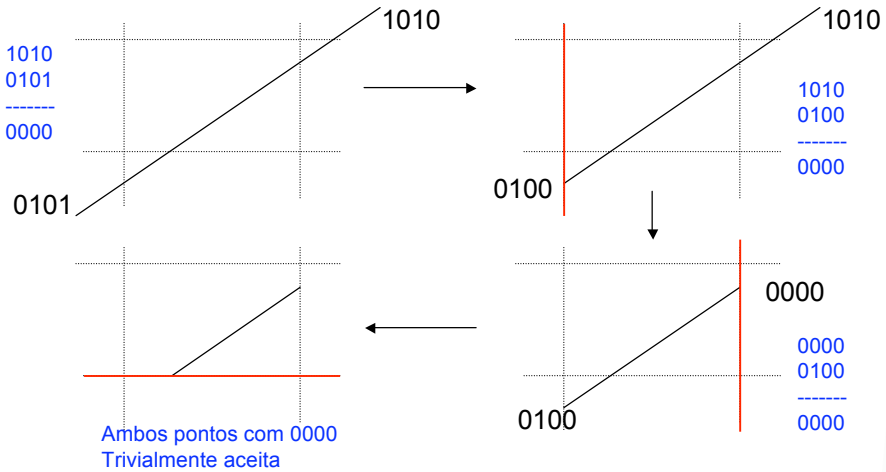
• O que fazer com os que não são nem aceitos nem recusados? Recorte por segmentos

Passos

1. Pontos são verificados para aceite e rejeição trivial utilizando os *outcodes*
2. Caso a linha precise ser recortada, divide em segmentos pelo limite da janela
3. Recorte iterativo até que a linha passe o teste de trivialmente aceita ou trivialmente rejeitada

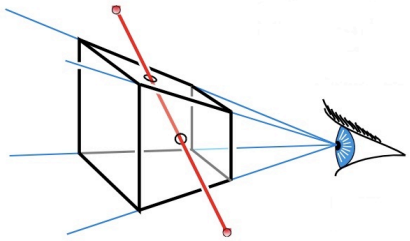
Cohen-Sutherland

Exemplo (ordem arestas l,r,b,t)



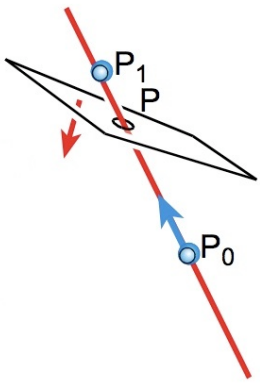
Recorte 3D

- Extensão de Cohen-Sutherland para 3D
- 6 outcodes ao invés de 4
  - bit 1: ponto está acima do volume
  - bit 2: ponto está abaixo do volume
  - bit 3: ponto está à direita do volume
  - bit 4: ponto está à esquerda do volume
  - bit 5: ponto está atrás do volume
  - bit 6: ponto está à frente do volume
- Os casos permanecem os mesmos



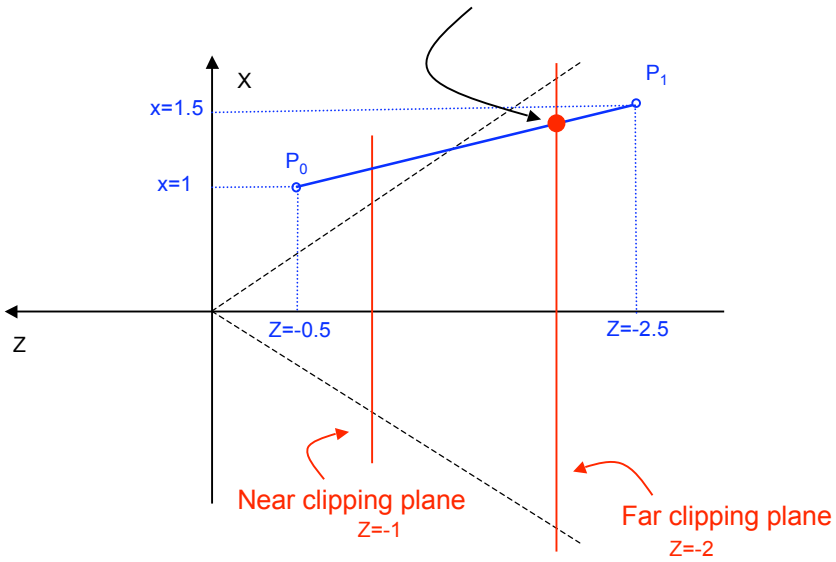
Calculando as intersecções

- Utilizamos a representação paramétrica de uma reta que passa por 2 pontos P_0 e P_1
- $L(t) = (1 - t) . P_0 + t . P_1$
- Substituimos esta equação na equação do plano que estamos recortando contra (genericamente $Ax+By+Cz+D=0$)



Exemplo

Qual o ponto de intersecção entre o segmento de reta dado por P_0P_1 e o plano z=-2?



- Eq paramétrica da reta
  - $L(t) = (1 - t)\ P_0 + t\ P_1$
  - $L_x(t) = (1 - t)\ 1 + 1.5\ t$ *
  - $L_z(t) = (1 - t)\ (-0.5) + -2.5t$ **
- Eq do plano z = -2
- Subs eq do plano em ** temos:

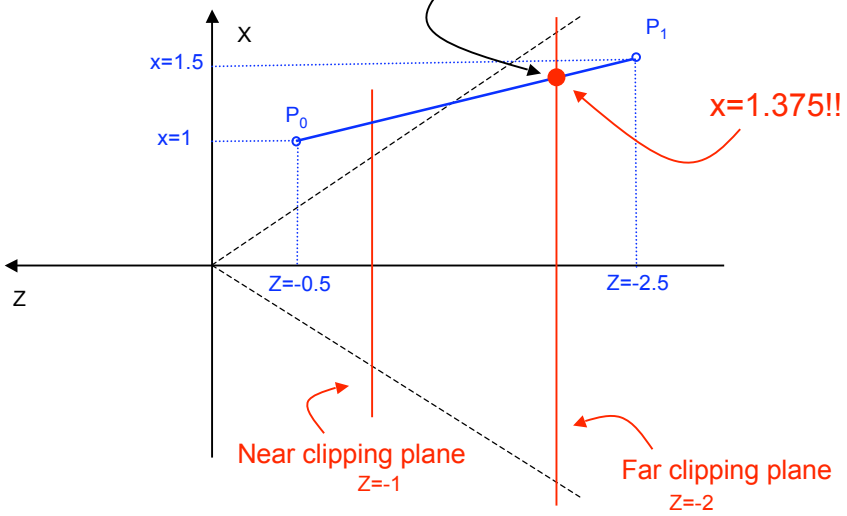
  $-2 = -0.5 + 0.5t - 2.5t$

  $t = 1.5/2 = 0.75$
- Fazendo t=0.75 em * temos

  $L_x(0.75) = (1-0.75)\ 1 + 1.5\ (0.75) = 1.375$

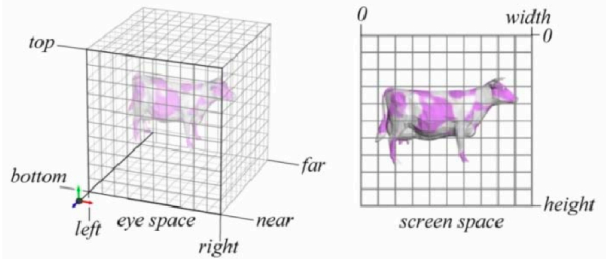
Exemplo

Qual o ponto de intersecção entre o segmento de reta dado por P_0P_1 e o plano z=-2?



Resumo

- Transformações Modelagem
- Iluminação (*Shading*)
- Transformação Câmera
- Recorte
- **Projeção**
- Rasterização
- Visibilidade

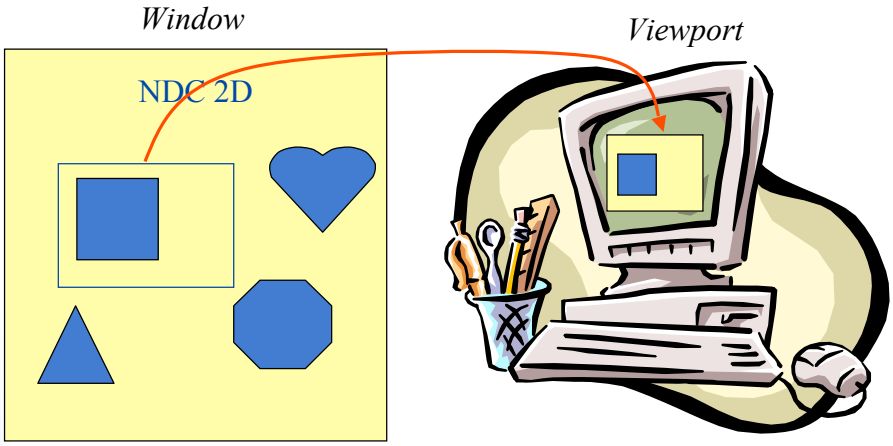


NDC

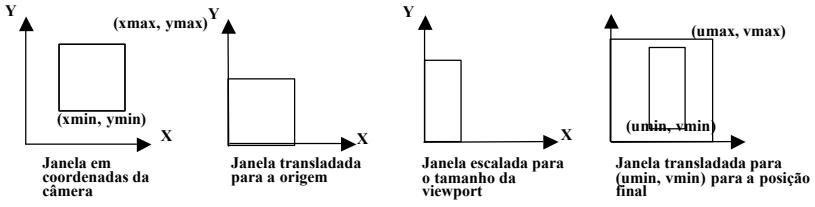
Coordenadas de Tela

✓Os vértices são projetados para coordenadas de tela

Onde estamos na tela?



Transformação Window-to-Viewport



translação → escala → translação

Transformações Modelagem

Iluminação (*Shading*)

Transformação Câmera

Recorte

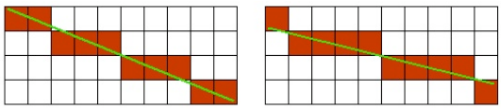
Map. Tela

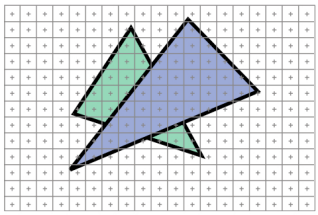
Rasterização

Visibilidade

✓Rasterização das linhas em pixels e possível preenchimento

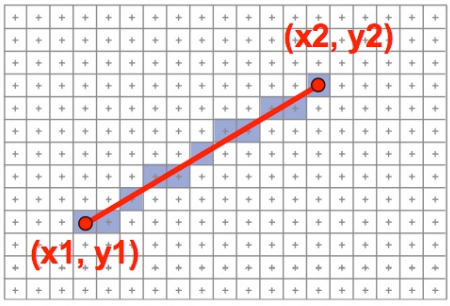
✓Interpolação dos valores (profundidade, normal, cor) dos vértices conforme necessidade





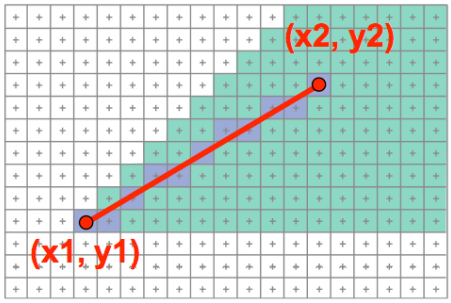
Algoritmos de rasterização

- *Objetivo*
  Aproximar primitivas matemáticas descritas através de vértices em *coordenadas reais* por meio de um conjunto de pixels em *coordenadas inteiras*



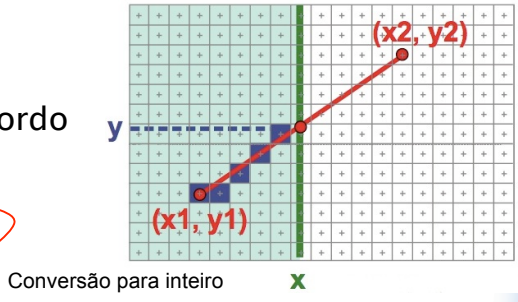
Algoritmos de rasterização

- Assumiremos
  $m = dy/dx$
  $0 < m <= 1$
- Exatamente um pixel por coluna



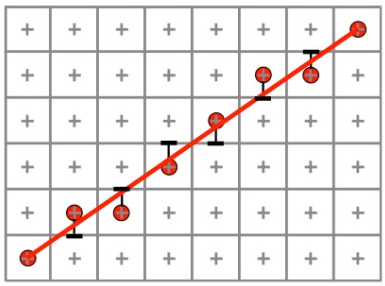
Algoritmo Incremental

- Calcula y em função de x
  - X sempre avança 1
  - Y acompanha de acordo com eq da reta
  - $y = y_1 + m\ (x - x_1)$
  - $y = y_1 + m$
  - $y_{tela} = \text{floor}(y+0.5)$



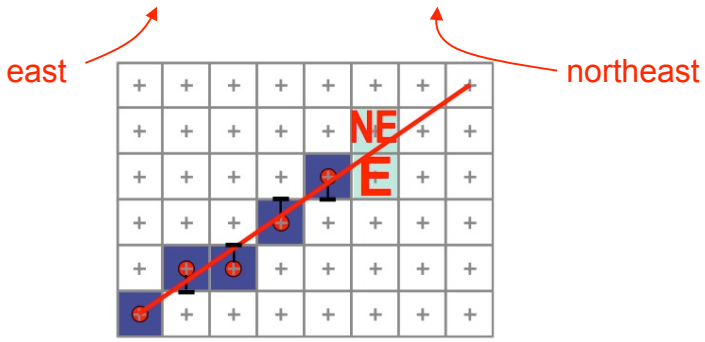
Algoritmo do Ponto Médio

- Bresenham – 1962
  - Atrativo porque usa somente operações aritméticas (não usa *round* ou *floor*)
  - Obtém mesma seqüência do algoritmo anterior
  - É incremental (usa o último resultado)



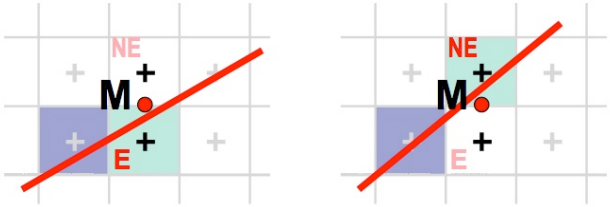
Bresenham
Algoritmo do Ponto Médio

- Observação chave
  Se estivermos no pixel (x_i,y_i) o próximo pixel ou é E (x_i+1, y_i) ou NE (x_i+1,y_i+1)



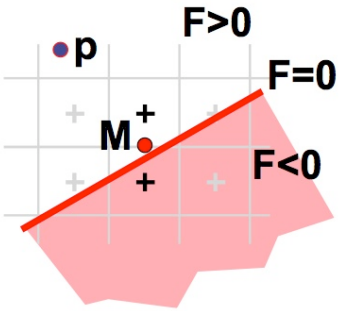
Qual pixel escolher? E ou NE?

- E se o segmento passar *abaixo* ou *sobre* o ponto médio M
- NE se o segmento passar *acima* do ponto médio M



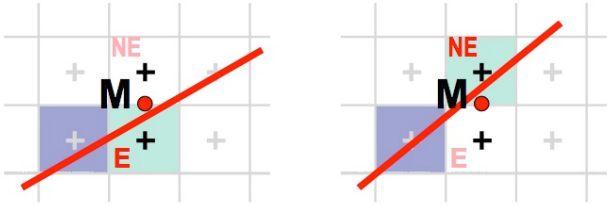
Como saber onde passa o segmento?

- Eq. da reta
  y = mx + b → y–mx–b = 0
  F(x,y) = y – mx – b
- O valor de F(x,y) nos diz onde o ponto está em relação à reta verticalmente
  - F(x,y) = 0 → Ponto está NA linha
  - F(x,y) > 0 → Ponto está ACIMA da linha
  - F(x,y) < 0 → Ponto está ABAIXO da linha



Decision Function

- Qual as coordenadas do ponto médio?
  (x+1, y + 0.5)
- D(x,y) = y – mx – b
  - D(x,y) < 0 → escolhe NE
  - D(x,y) > 0 → escolhe E
  - D(x,y) = 0 → arbitrariamente escolhe uma possibilidade (consistentemente)



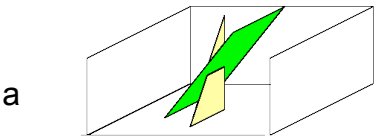
Outros tipos de retas

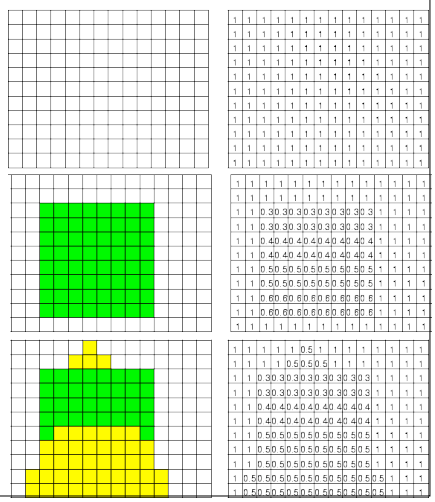
- Caso for horizontal ou vertical tratar como caso especial
- Caso $x_1 > x_2$ inverter ordem dos pontos
- Para as outras inclinações de segmentos generalização do caso visto (não veremos aqui...)

Resumo

- Transformações Modelagem
- Iluminação (*Shading*)
- Transformação Câmera
- Recorte
- Projeção
- Rasterização
- **Visibilidade**

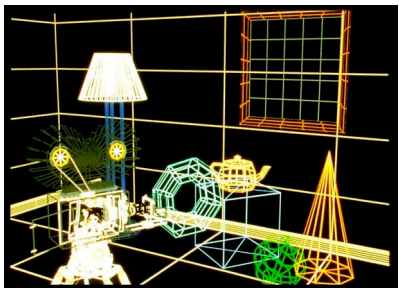
✓Z-buffer: cada pixel armazena a coordenada z mais próxima

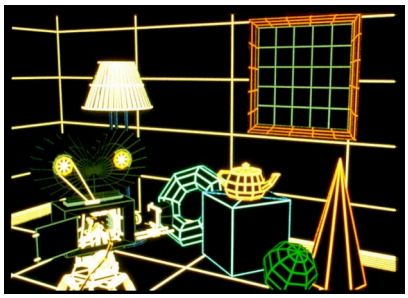




- Transformações Modelagem
- Iluminação (*Shading*)
- Transformação Câmera
- Recorte
- Projeção
- Rasterização
- **Visibilidade**

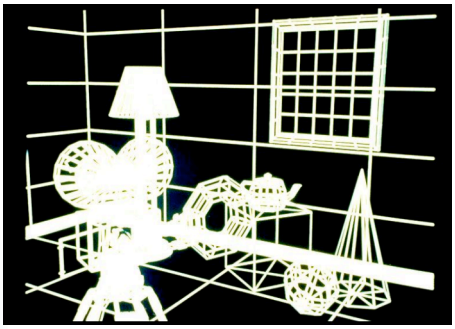
✓Desenhar apenas os polígonos visíveis

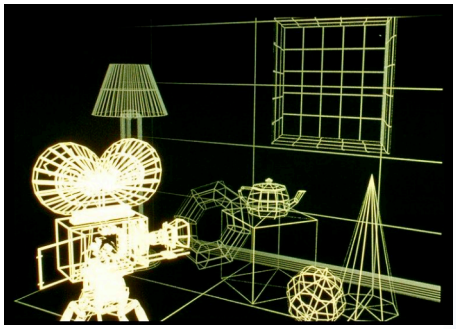




Objetivo Principal

Aumento do realismo





O que foi feito aqui?

Duas Abordagens

- A primeira abordagem determina quais dos ***n*** objetos são visíveis em cada pixel da imagem (precisão de imagem)

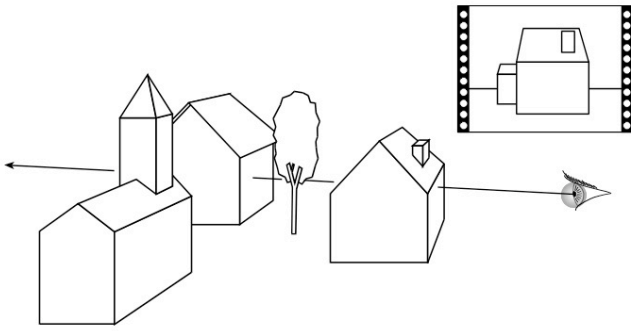
```
para( cada pixel na imagem)
{
          determinar o objeto mais próximo do observador;
          desenhar o pixel na cor apropriada;
}
```

- A segunda abordagem compara objetos diretamente uns com os outros, eliminando objetos inteiros ou porções deles que não são visíveis (precisão de objeto)

```
para( cada objeto no mundo)
{
          determinar as partes do objeto cuja visão não está obstruída
                    por outras partes dele ou de qualquer outro objeto;
          desenhar essas partes na cor apropriada;
}
```

Primeira Abordagem

- Ray Casting*



*veremos com mais detalhes nas próximas aulas

Segunda Abordagem

- Mais genérica e resolve o problema de forma exata, independentemente da resolução da imagem
- Dados n objetos complexidade n^2
- Testar cada objeto com todos os outros

```
para( cada objeto no mundo)
{
          determinar as partes do objeto cuja visão não está obstruída
                    por outras partes dele ou de qualquer outro objeto;
          desenhar essas partes na cor apropriada;
}
```

Z-Buffer
Depth Buffer

- Criado por Catmull (1974)
- Idéia:
  - Na etapa de rasterização, considerar z, além de x e y
  - Além do *framebuffer*, teremos um *depth buffer* (z buffer), onde estarão armazenados os z's
  - Inicialmente, setamos todo o z-buffer para infinito (ou para o *far clipping plane*)
  - *Framebuffer* é todo setado para cor de fundo

Z-Buffer

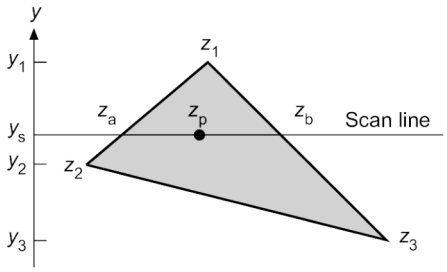
- Realizar a etapa de rasterização, usando a seguinte rotina para preencher o *framebuffer*

```
WritePixel(int x, int y, float z, colour)
if ( z < zbuf[x][y] ) then
    zbuf[x][y] = z;
    frambuffer[x][y] = colour;
end
```

- Ou seja, o buffer armazena o pixel mais próximo até então em (x,y)
- Não renderiza um pixel se ele tem a profundidade maior que z[x][y]

Z-Buffer

- Determinando profundidades



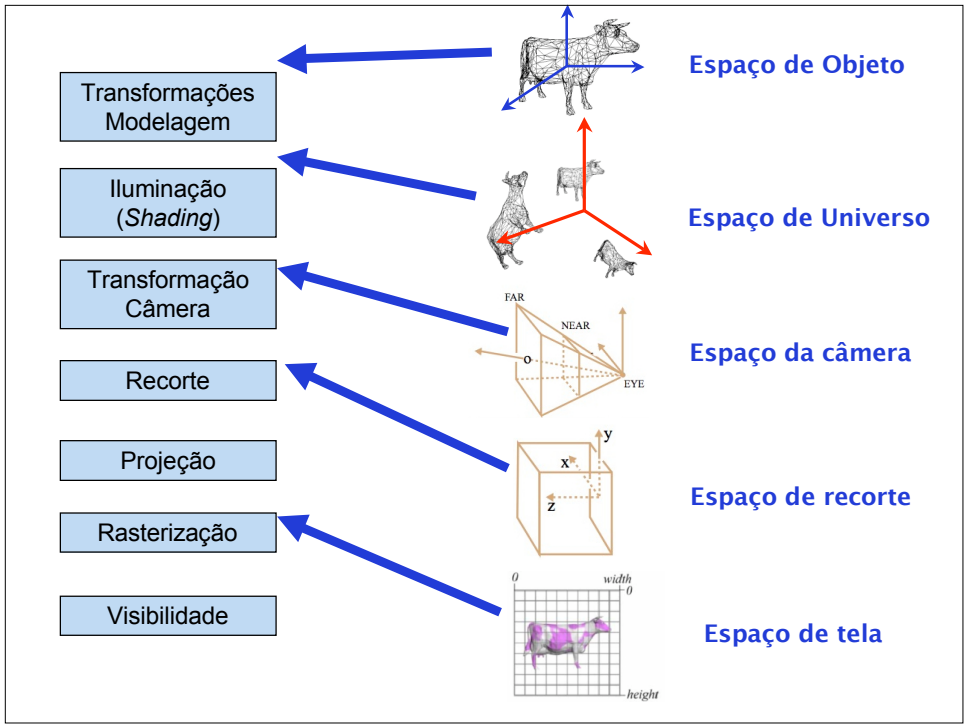
$$z_a = z_1 - (z_1 - z_2)\frac{y_1 - y_s}{y_1 - y_2}$$

$$z_b = z_1 - (z_1 - z_3)\frac{y_1 - y_s}{y_1 - y_3}$$

$$z_p = z_b - (z_b - z_a)\frac{x_b - x_p}{x_b - x_a}$$

Z-Buffer

- Vantagens
  - Fácil de implementar
  - Simples de implementar em hardware
  - Objetos não precisam ser necessariamente polígonos
- Desvantagens
  - Consome muita memória
    - Se diminuir memória, podem ocorrer erros
      - faces A = 0.89485, B = 0.89501 / Round: A = 0.895, B = 0.895
      - Renderiza A, depois B (Fica B, errado!, pois A é mais próxima)
  - Dependente do dispositivo
  - O algoritmo pode desenhar um pixel muitas vezes


Transformações Modelagem
Iluminação (*Shading*)
Transformação Câmera
Recorte
Projeção
Rasterização
Visibilidade
Espaço de Objeto
Espaço de Universo
Espaço da câmera
Espaço de recorte
Espaço de tela
FAR
NEAR
EYE
o
x
y
z
0
width
height