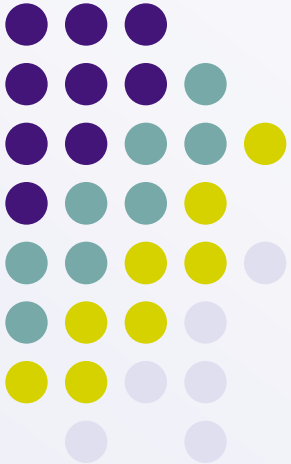
Universidade de São Paulo
Instituto de Matemática e Estatística

Introdução à Computação Gráfica

Marcel P. Jackowski
mjack@ime.usp.br

Aula #13



Objetivos

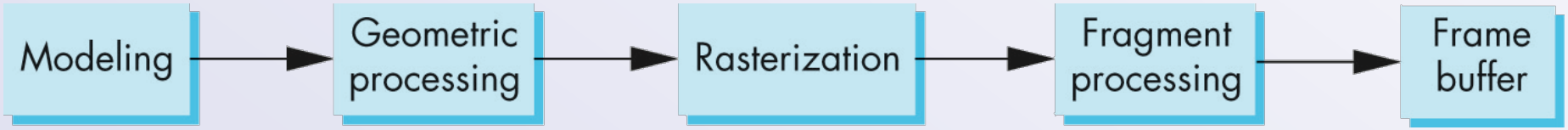
- Técnicas de implementação
  - Recortes de linhas (retas) e polígonos
  - Remoção de superfícies escondidas

Visão geral

- Ao final do pipeline de geometria, vértices foram agrupados em primitivas.
- Primitivas que estão localizadas fora (ou parcialmente fora) da caixa de visualização devem ser removidas (recortadas)
- Devemos determinar quais pixels são afetados por cada primitiva:
  - Geração de fragmentos
  - Rasterização

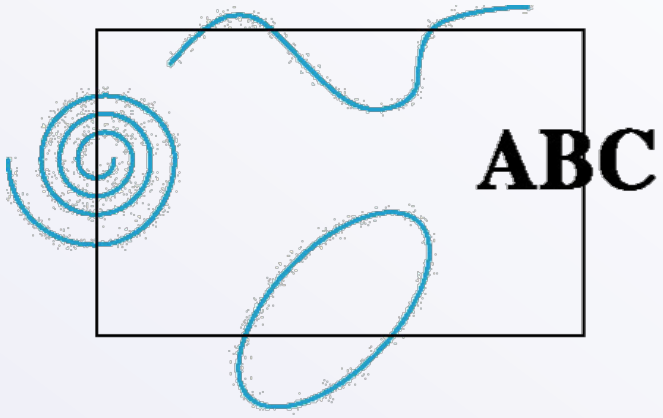
Pipeline de Geometria

- Transformações
- Recortes
- Rasterização
- Certas tarefas são adiadas até a etapa de processamento de fragmentos:
  - Remoção de superfícies escondidas
  - Antiserrilhamento (antialiasing)



Recorte (“Clipping”)

- Em 2D, utiliza-se uma “janela de recorte”, enquanto em 3D, usamos um “volume de recorte”
- Fácil recortar polígonos baseados em segmentos de reta
- Porém mais difícil recortar curvas e textos:
  - Converter antes para linhas e polígonos



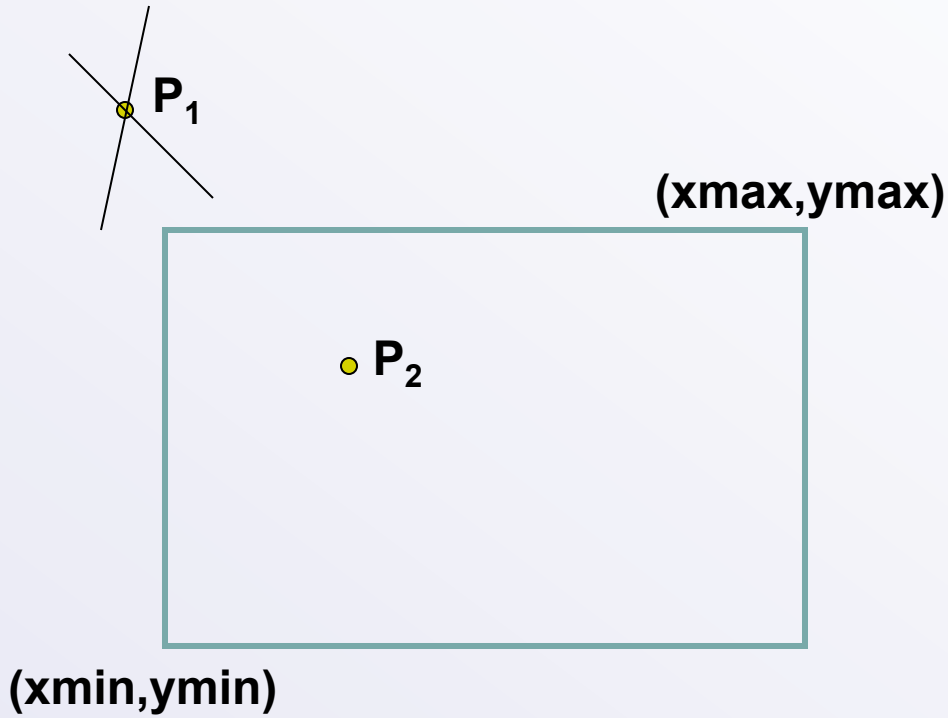
Tipos de Recorte

- Recorte de pontos
- Recorte de retas
  - Algoritmo de Cohen-Sutherland
  - Algoritmo de Liang-Barsky
- Recorte de polígonos
  - Algoritmo de Sutherland-Hodgman

Recorte de pontos

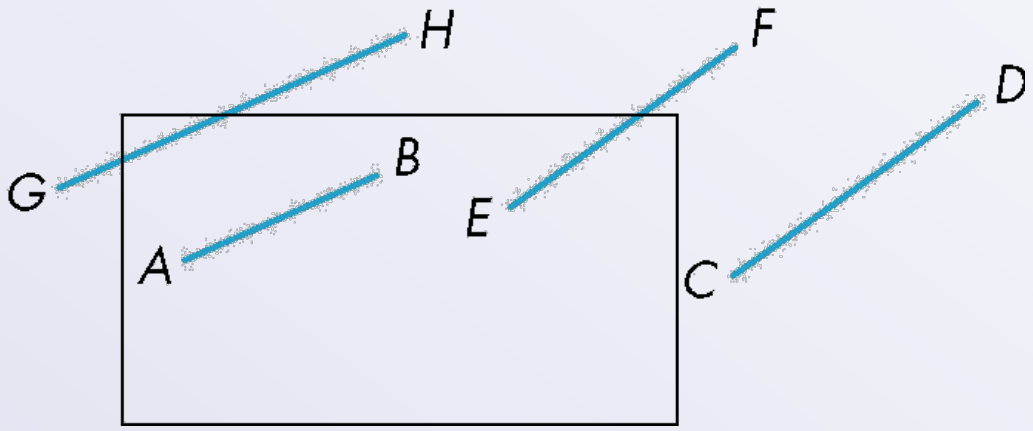
```
if x >= xmin AND
   x <= xmax AND
   y >= ymin AND
   y <= ymax

  // Ponto DENTRO,
     desenha
else
  // Ponto FORA,
     rejeita
```



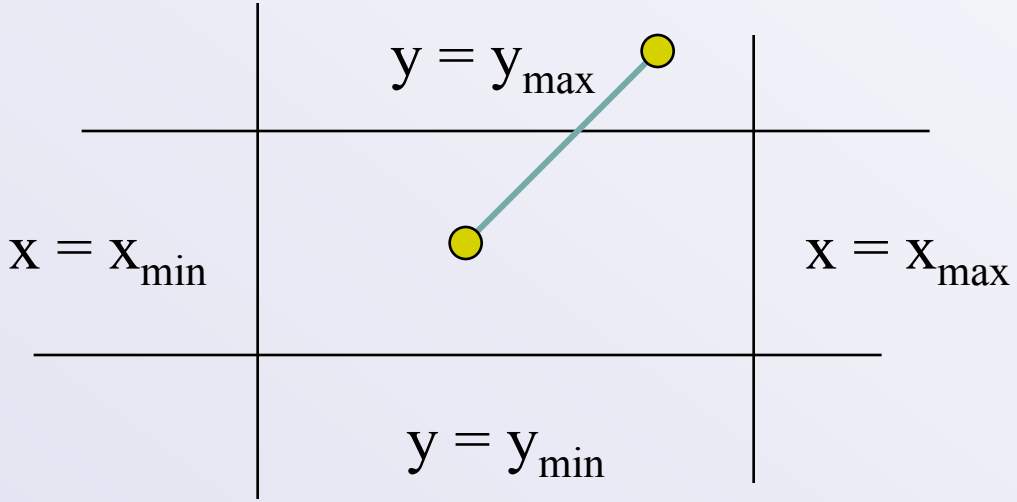
Recorte de retas em 2D

- Para cada reta, calcular as interseções com cada um dos lados da janela de recorte;
  - Ineficiente, pois requer cálculo de várias divisões.



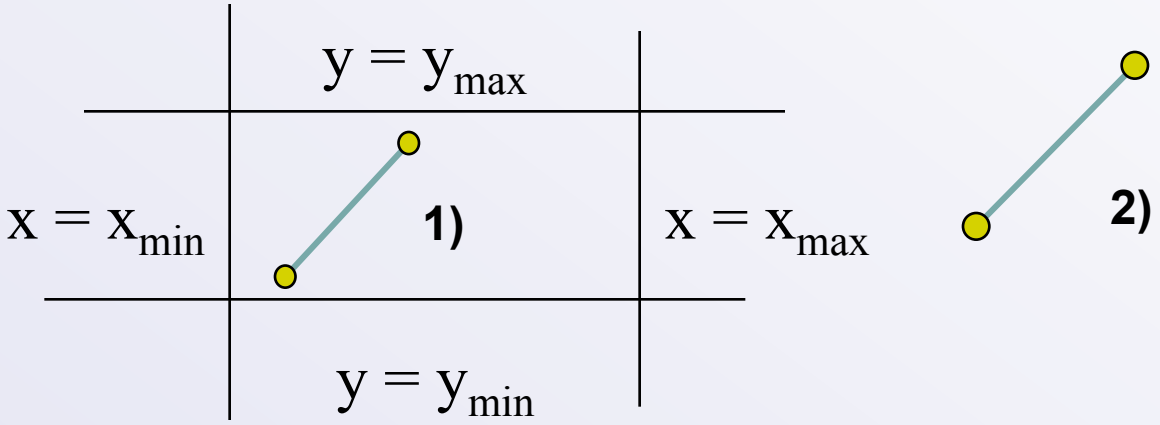
Algoritmo de Cohen-Sutherland

- Eliminar o máximo número de possibilidades possíveis sem calcular interseções.
- Começa com quatro retas suporte que determinam os lados da janela de recorte.



Possibilidades

1) Se os pontos extremos do segmento de reta estão contidos dentro das quatro retas suporte, então **desenha (aceita)** o segmento de reta.



2) Se os pontos finais estão fora de todas as retas suporte, mas estão no mesmo lado de uma delas, então **descarta (rejeita)** o segmento.

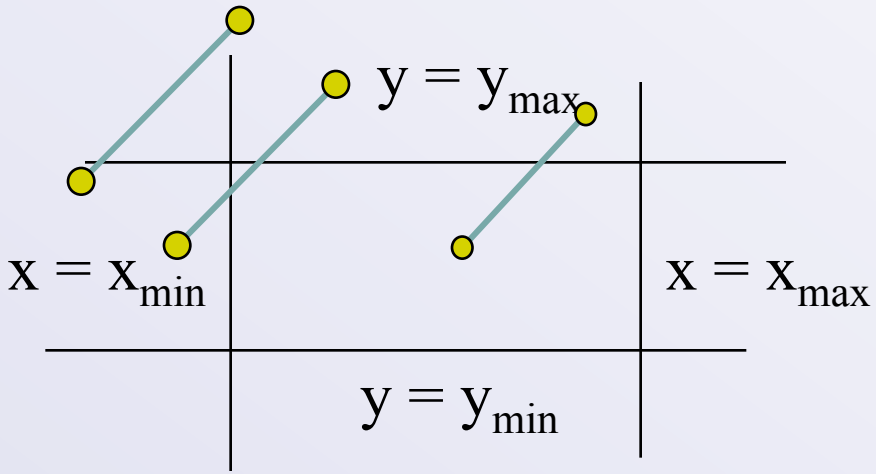
Possibilidades

3) Um ponto dentro e um ponto fora

- Requer o cálculo de pelo menos uma interseção

4) Ambos pontos extremos fora

- Parte da reta pode estar dentro da janela
- Calcular pelo menos uma interseção

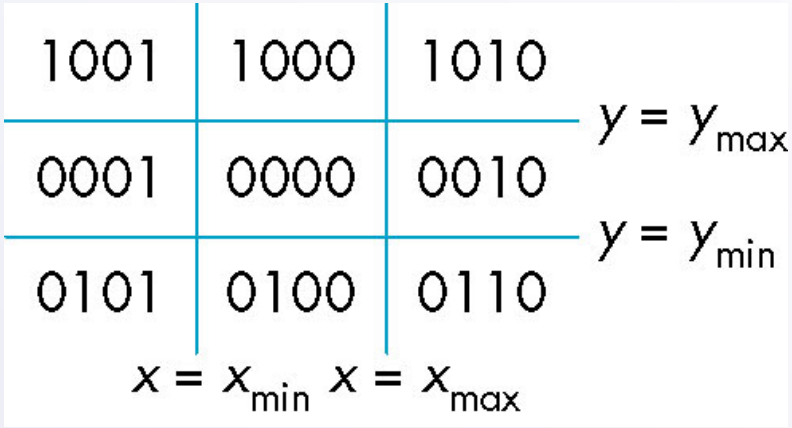


Códigos

- Para cada ponto, associamos um código de resultado

$$b_0 b_1 b_2 b_3$$

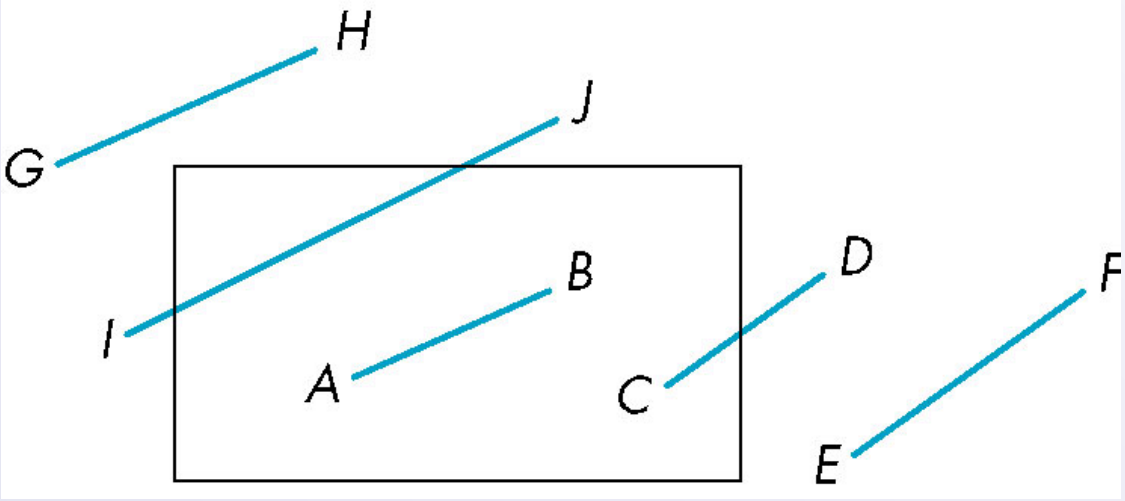
$b_0 = 1$ se $y > y_{max}$, 0 contrário
$b_1 = 1$ se $y < y_{min}$, 0 contrário
$b_2 = 1$ se $x > x_{max}$, 0 contrário
$b_3 = 1$ se $x < x_{min}$, 0 contrário



- É *visível* se os códigos dos dois pontos extremos **são** 0000
- É *invisível* se o AND lógico dos códigos **não é** 0000
- É *candidato* ao recorte se o AND lógico dos códigos dos pontos terminais **é** 0000

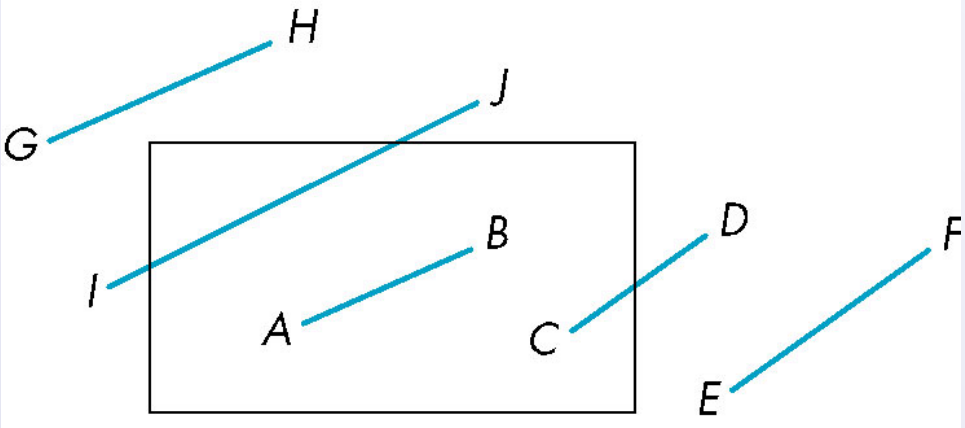
Interpretando os códigos

- Segmento AB:
  - Código(A) = Código(B) = 0
  - Aceita segmento de reta



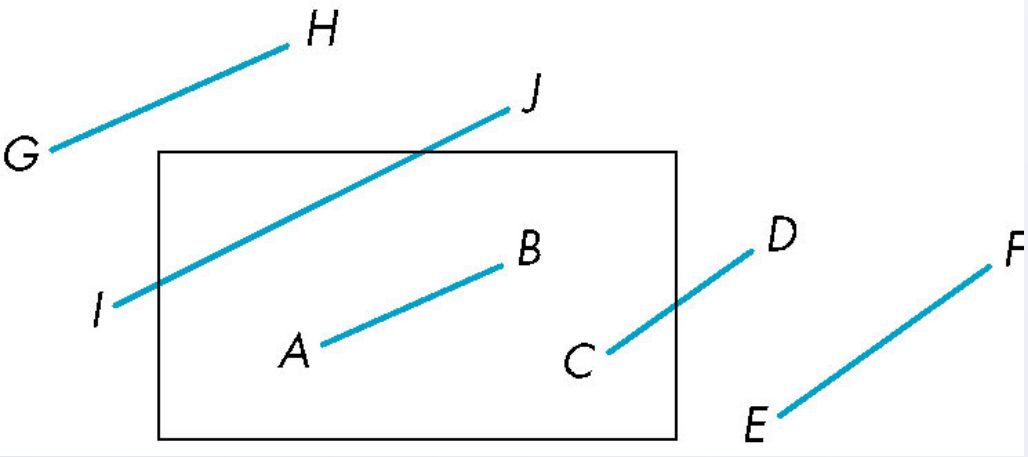
Interpretando os códigos

- Segmento CD:
  - Código (C) = 0, mas Código(D) != 0
  - Calcula intersecção
  - A posição do 1 em Código(D) determina a aresta de interseção



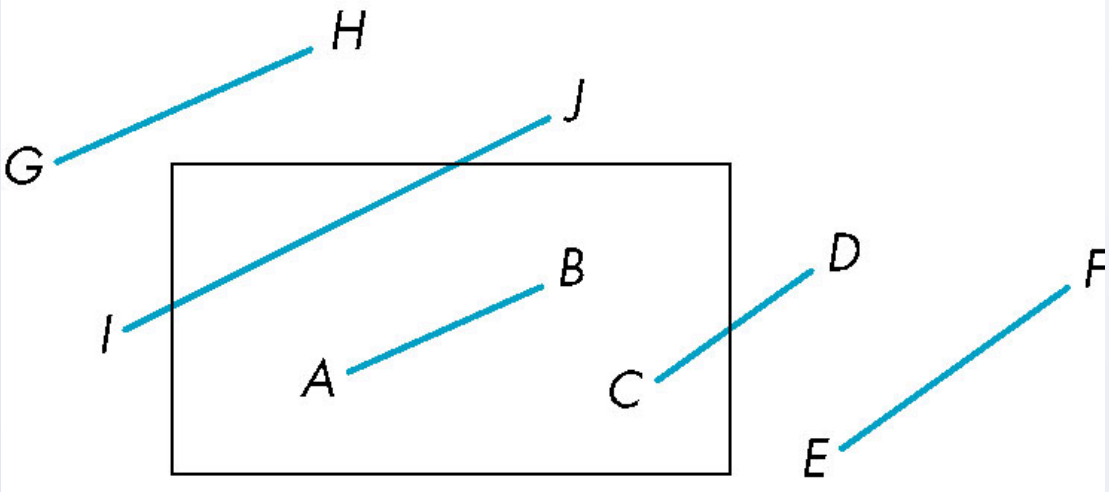
Interpretando os códigos

- Segmento EF:
  - Código(E) AND Código(F) != 0
  - Ambos códigos possuem 1 no mesmo bit
  - O segmento está fora da janela de recorte
  - Rejeita



Interpretando os códigos

- Segmentos GH e IJ:
  - A operação lógica AND resulta em 0
  - Calcular interseção com um dos lados da janela
  - Calcular novo código do ponto de intersecção
  - Re-executa algoritmo

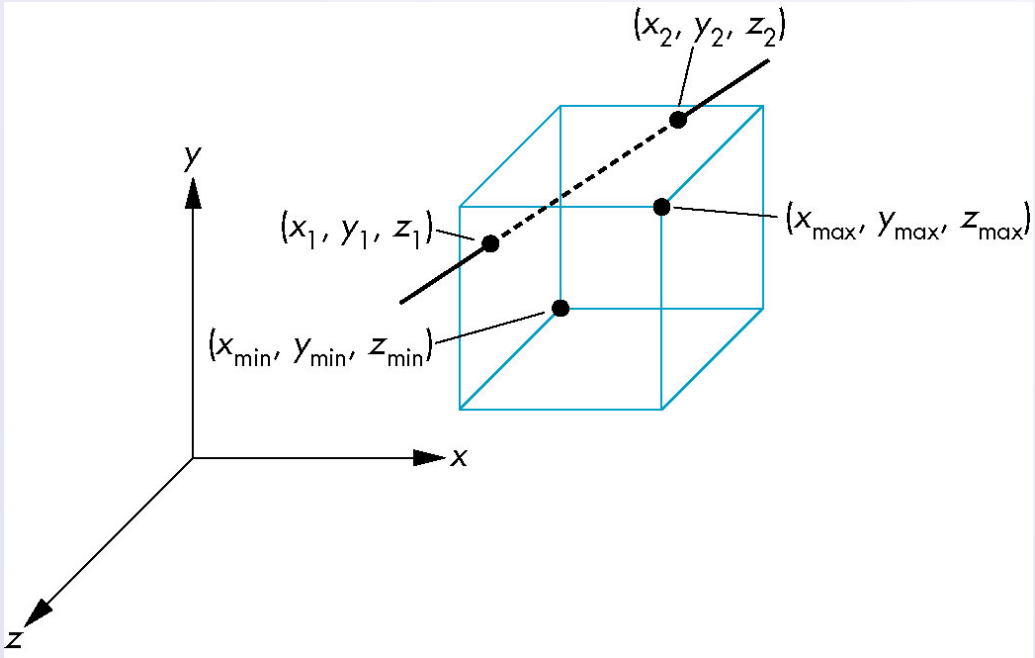


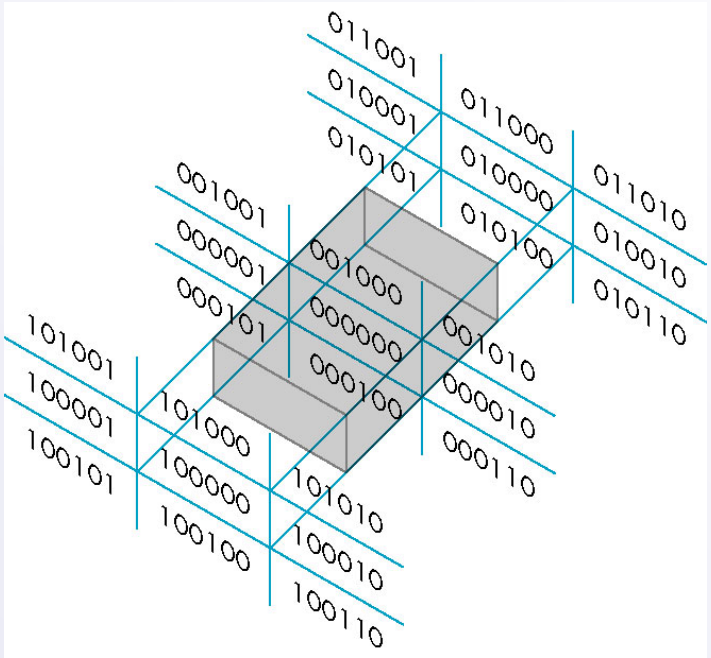
Eficiência

- Em muitas aplicações, a janela de recorte é pequena em relação ao tamanho da base de dados
  - A maioria dos segmentos de reta estarão em um ou mais lados da janela de recorte e podem ser eliminados observando os códigos de resultado;
- Se torna ineficiente quando temos que re-executar o algoritmo no caso de múltiplas interseções.

Cohen-Sutherland em 3D

- Usa códigos de 6 bits





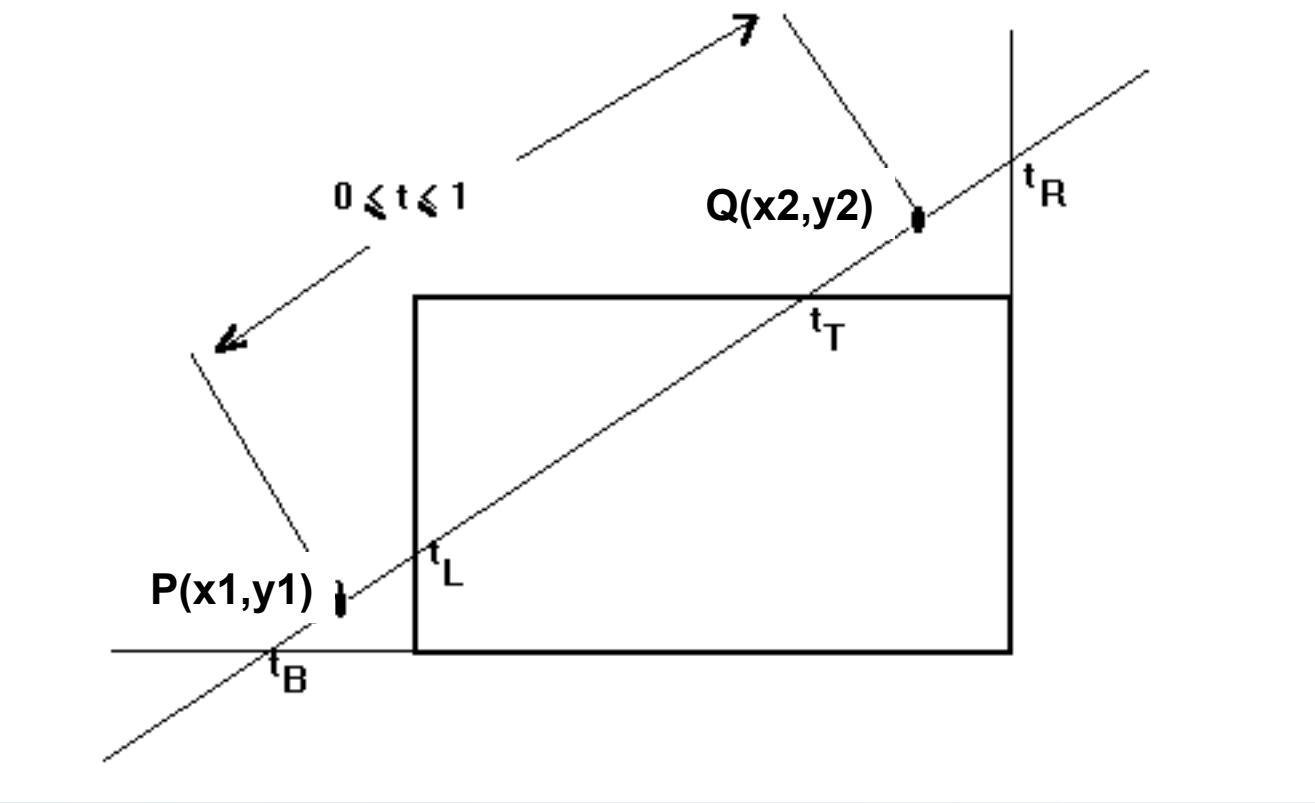
Algoritmo de Liang-Barsky

- Refinamento que consiste em representar a reta de forma paramétrica
- Porção da reta em 2D não recortada deve satisfazer:

$$x_{\min} \le x_1 + t\,\Delta \mathrm{x} \le x_{\max} \qquad \Delta x = x_2 - x_1$$

$$y_{\min} \le y_1 + t\,\Delta \mathrm{y} \le y_{\max} \qquad \Delta y = y_2 - y_1$$

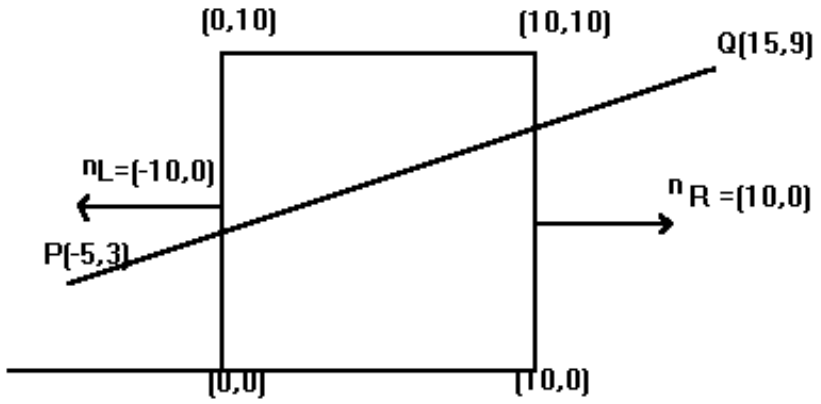
Liang-Barsky



$$x = x_1 + (x_2 - x_1)*t = x_1 + dx*t$$

$$y = y_1 + (y_2 - y_1)*t = y_1 + dy*t$$

Exemplo 1



$t_B = \frac{0-3}{9-3} = \frac{-1}{2}$ t < 0 then ignore

$t_L = \frac{0-(-5)}{15-(-5)} = 1/4$

$t_R = \frac{10-(-5)}{15-(-5)} = 3/4$

$t_T = \frac{10-3}{9-3} = 7/6$ t > 1 then ignore

The next step we consider if tvalue is entering or exiting by using inner product.

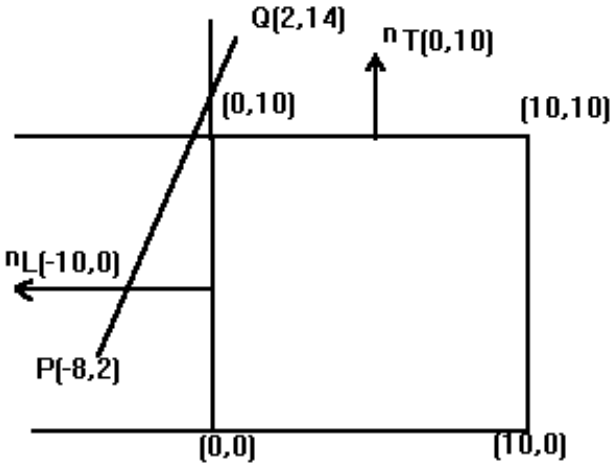
(Q-P) = (15+5,9-3) = (20,6)

At left edge (Q-P)nL = (20,6)(-10,0) = -200 < 0 **entering** so we set tmin = 1/4

At right edge (Q-P)nR = (20,6)(10,0) = 200 > 0 **exiting** so we set tmax = 3/4

Because $t_{min} < t_{max}$ then we draw a line from (-5+(20)*(1/4), 3+(6)*(1/4)) to (-5+(20)*(3/4), 3+(6)*(3/4))

Exemplo 2



$t_B = \frac{0-2}{14-2} = -1/6$ t < 0 then ignore

$t_T = \frac{10-2}{14-2} = 8/12$

$t_L = \frac{0-(-8)}{2-(-8)} = 8/10$

$t_R = \frac{10-(-8)}{2-(-8)} = 18/10$ t > 1 then ignore

The next step we consider if tvalue is entering or exiting by using inner product.

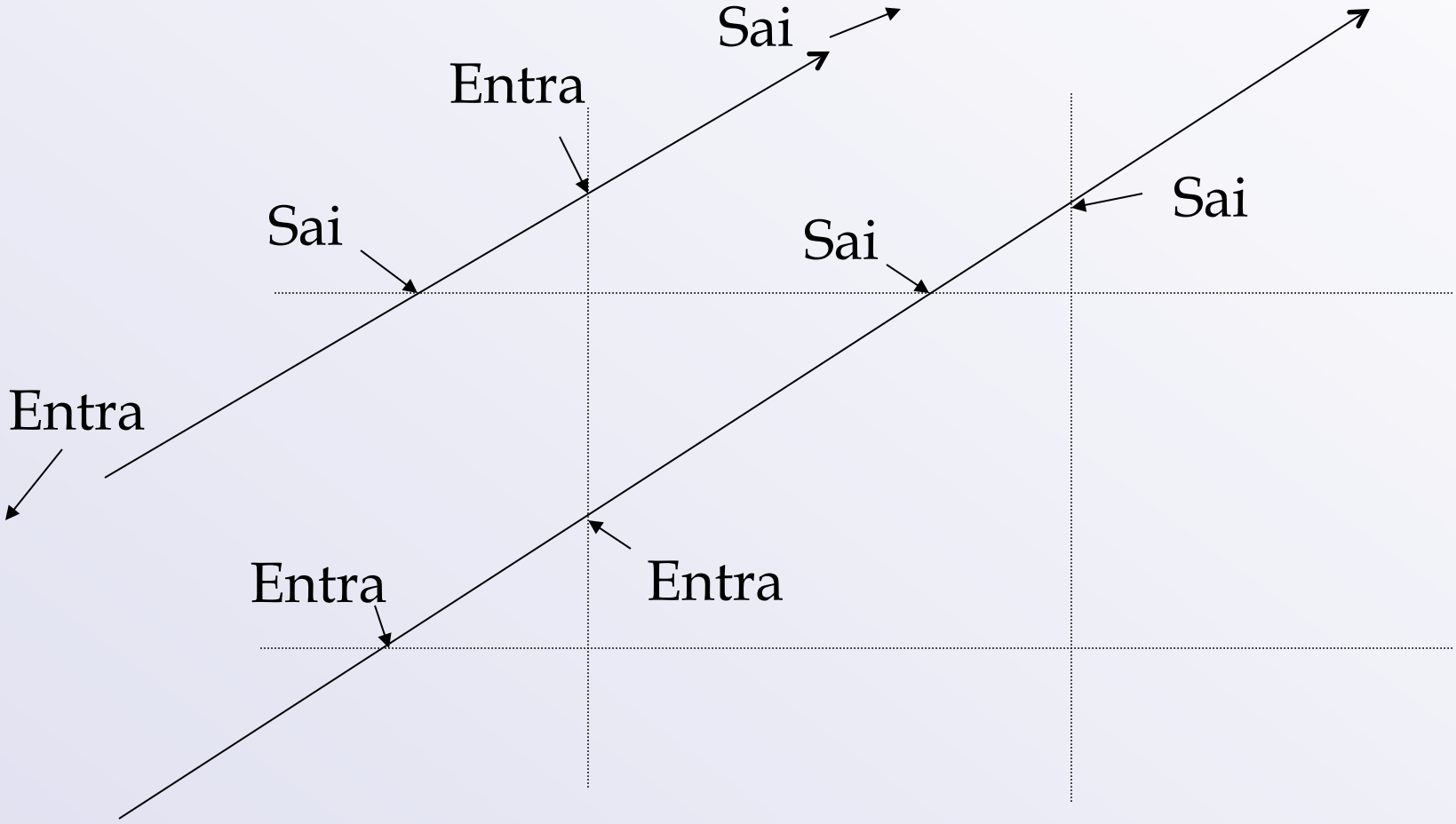
(Q-P) = (2+8,14-2) = (10,12)

At top edge (Q-P)nT = (10,12)(0,10) = 120 > 0 **exiting** so we set tmax = 8/12

At left edge (Q-P)nL = (10,12)(-10,0) = -100 < 0 **entering** so we set tmin = 8/10

Because tmin > tmax then we don't draw a line.

Algoritmo de Liang-Barsky

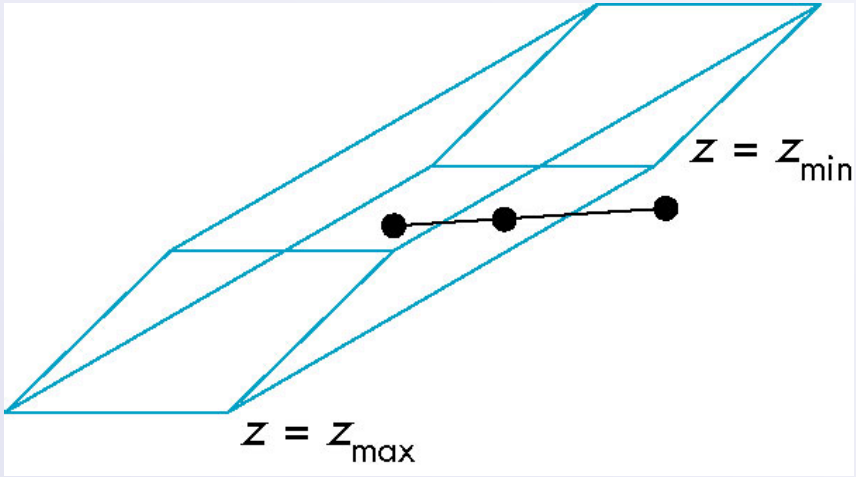


Pseudo-código

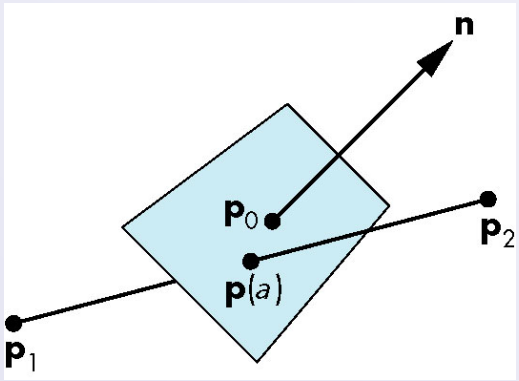
- Computar valores de *t* para os pontos de interseção
- Classificar pontos em **entra** ou **sai**
- Vértices do segmento recortado devem corresponder a dois valores de *t:*
  - t_{min}= max (0, *t*' s do tipo **entra**)
  - t_{max}= min (1, *t*' s do tipo **sai**)
- Se $t_{min}<t_{max}$, segmento recortado é não nulo
  - Computar vértices substituindo os valores de *t*
- Na verdade, o algoritmo calcula e classifica valores de *t* um a um.

Recorte e Normalização

- O recorte genérico em 3D requer o cálculo da intersecção de segmentos de reta em relação a um plano arbitrário.
- Exemplo: visão oblíqua



Intersecção entre Reta e Plano



$$\mathbf{p}(\alpha) = (1-\alpha)\mathbf{p}_1 + \alpha\mathbf{p}_2$$

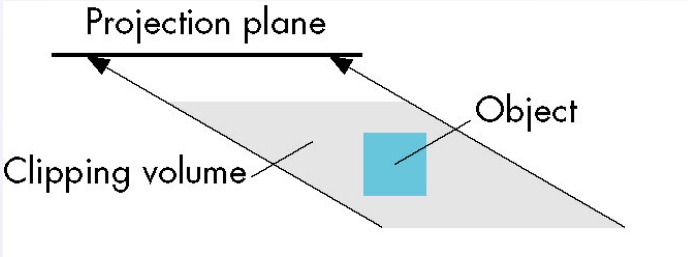
$$\mathbf{n}\cdot(\mathbf{p}(\alpha) - \mathbf{p}_0) = 0$$

$$\alpha = \frac{n\cdot(p_0 - p_1)}{n\cdot(p_2 - p_1)}$$

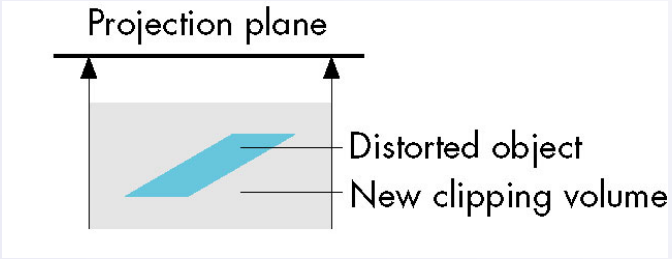
Normalização do Volume

- Após a normalização, recortamos em relação aos lados de um paralelepípedo;
- Uma intersecção típica somente requer uma subtração: $x > x_{max}$?

Visão superior



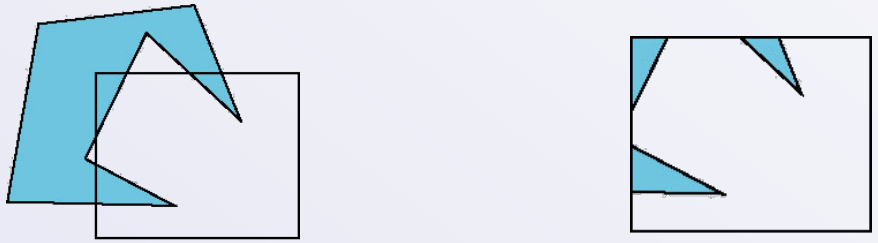
Antes da normalização



Após normalização

Recorte de Polígonos

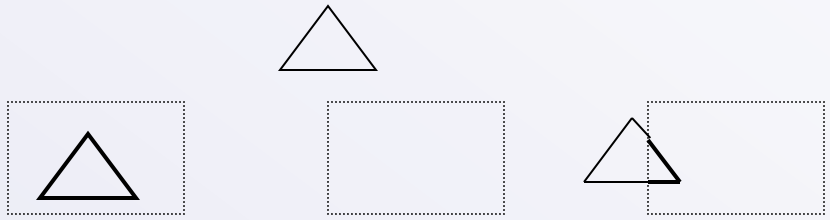
- Não tão simples quanto o recorte de linhas
  - O recorte de uma reta resulta em no máximo um outro segmento de reta
  - O recorte de um polígono pode resultar em múltiplos polígonos



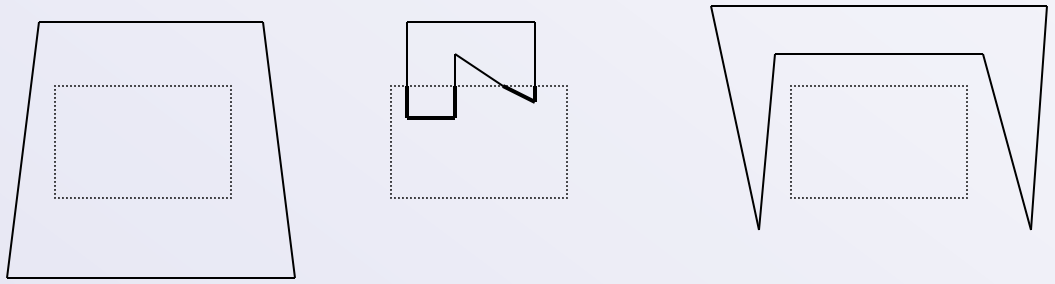
- O recorte de um polígono convexo pode resultar em no máximo mais um outro polígono.

Recorte de Polígono contra Retângulo

- Casos Simples



- Casos Complicados

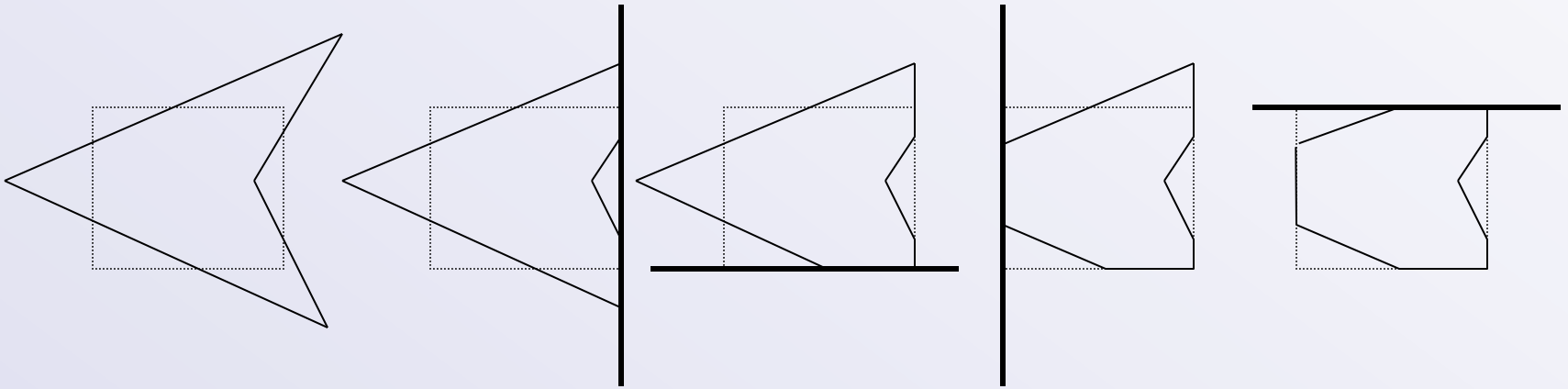


Recorte de Polígono contra Retângulo

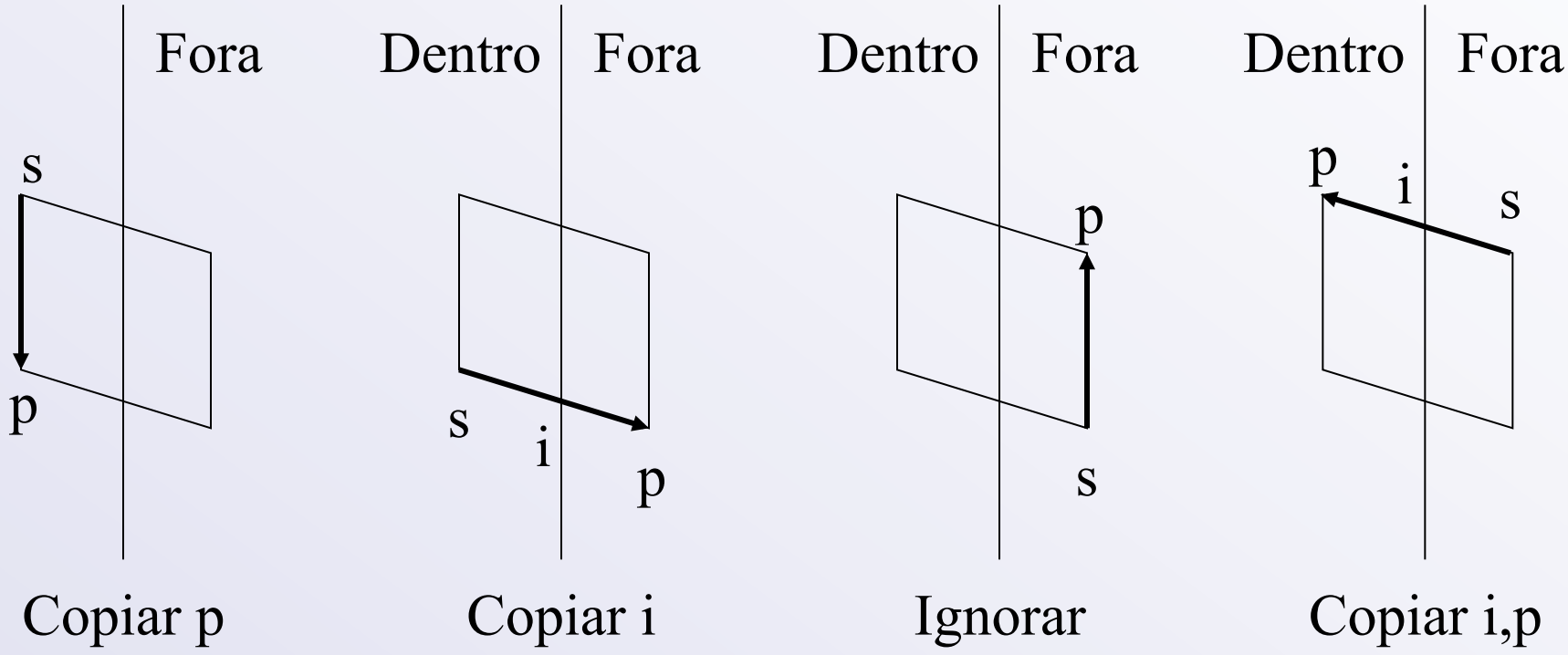
- Inclui o problema de recorte de segmentos de reta
  - Polígono resultante tem vértices que são
    - Vértices da janela,
    - Vértices do polígono original, ou
    - Pontos de interseção aresta do polígono/aresta da janela
- Dois algoritmos clássicos
  - Sutherland-Hodgman (1974)
    - Figura de recorte pode ser qualquer polígono convexo
  - Weiler-Atherton (1977)
    - Figura de recorte pode ser qualquer polígono

Algoritmo de Sutherland-Hodgman

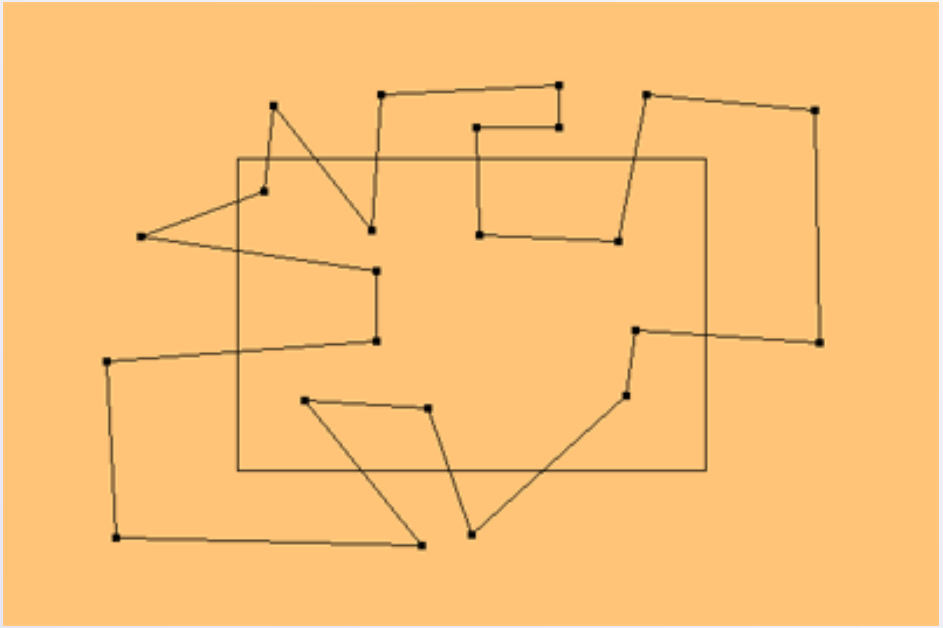
- Idéia é semelhante à do algoritmo de Sutherland-Cohen
  - Recortar o polígono sucessivamente contra todos os semi-espaços planos da figura de recorte

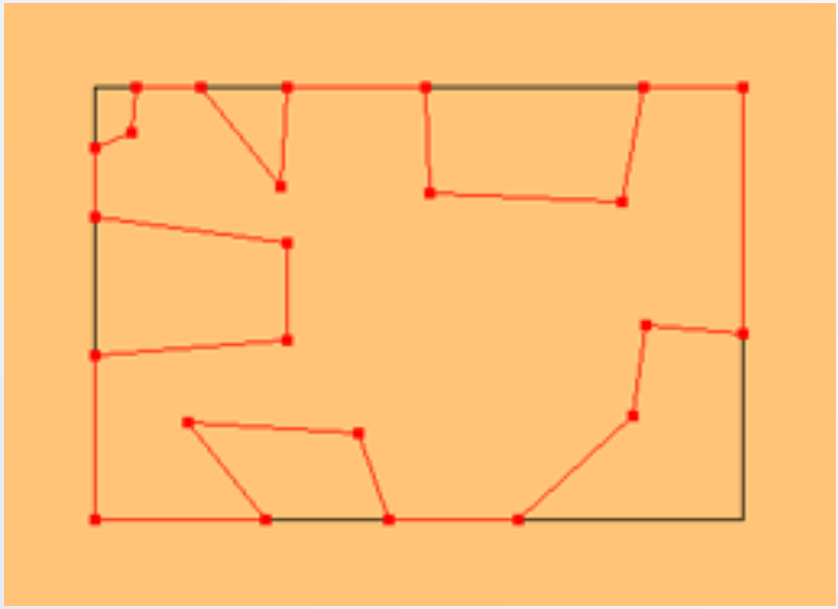


Algoritmo de Sutherland-Hodgman



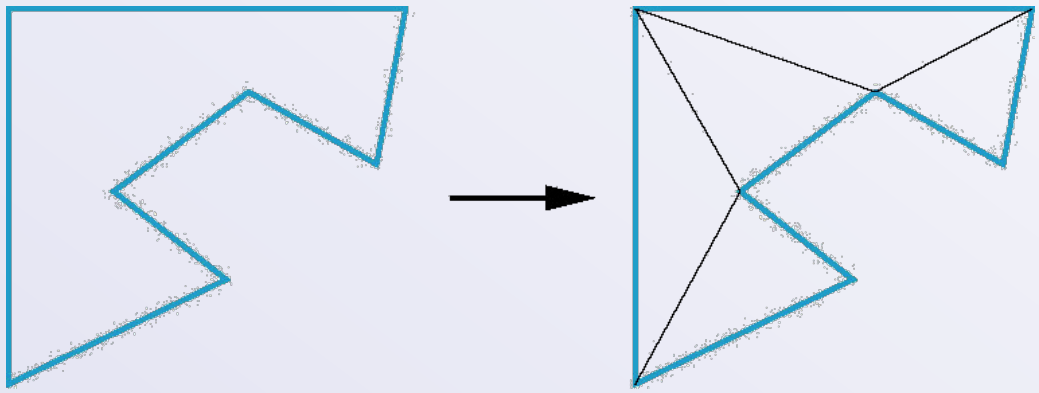
Sutherland-Hodgman – Exemplo





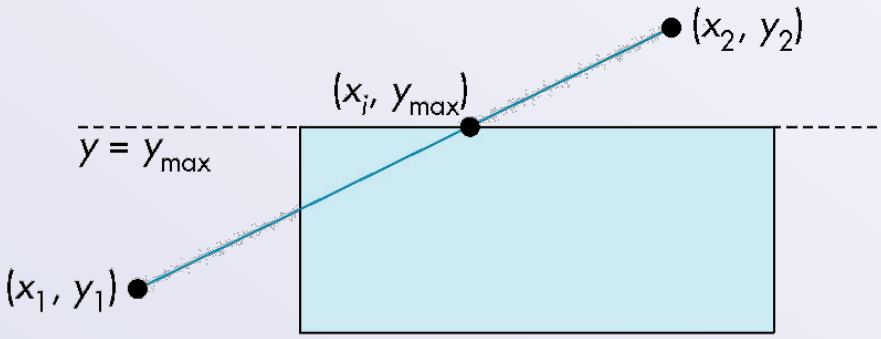
Tecelagem e convexidade

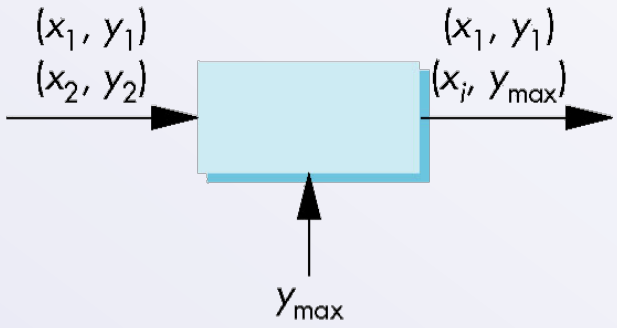
- Uma estratégia é substituir polígonos côncavos por um conjunto de polígonos triangulares (uma *tecelagem*)
- Facilita o preenchimento
- Código de tecelagem presente na GLU:
  - gluTess*



Recorte como uma caixa preta

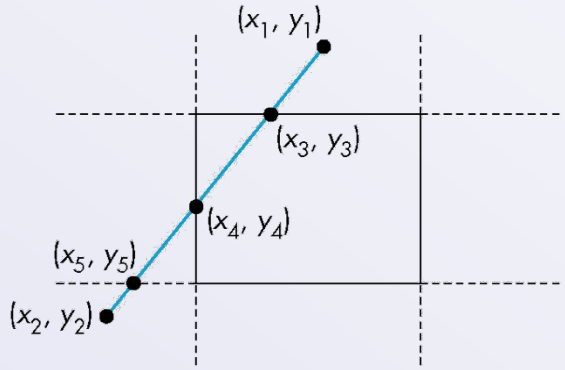
- Podemos considerar o recorte de linhas como uma função que recebe dois vértices e produz: 0 vértices ou os vértices de uma linha recortada

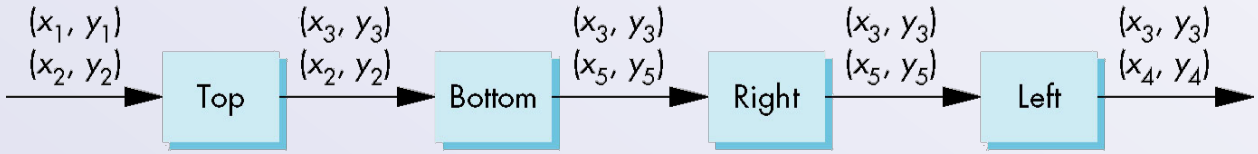




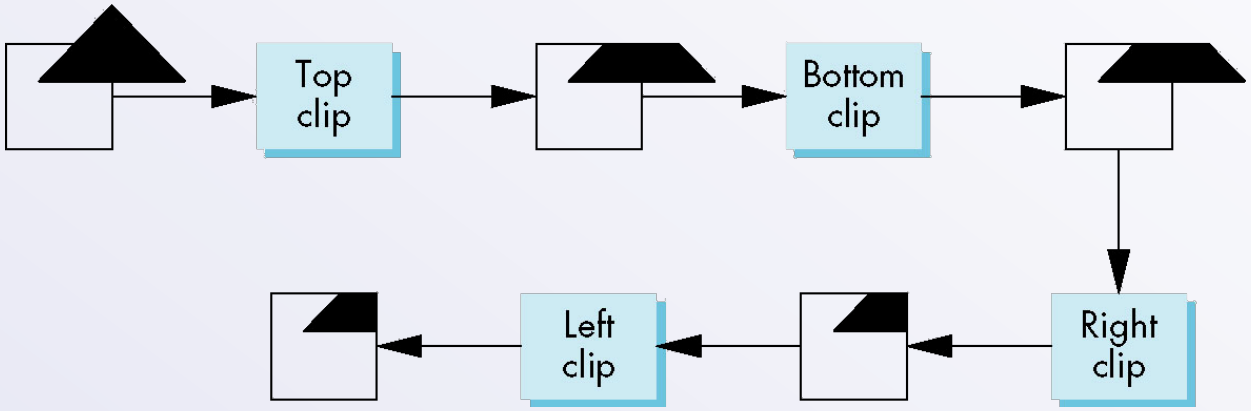
Recorte de linhas em pipeline

- O recorte em relação a cada lado da janela é independente dos outros lados
  - Podemos utilizar 4 recortes independentes em forma de pipeline





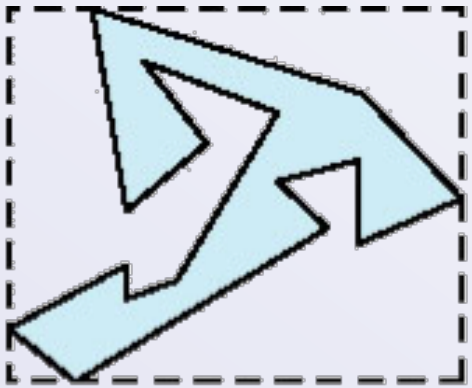
Recorte de polígonos em pipeline



- 3D: adicionar recortes frontal e traseiro
- Estratégia utilizada na SGI Geometry Engine

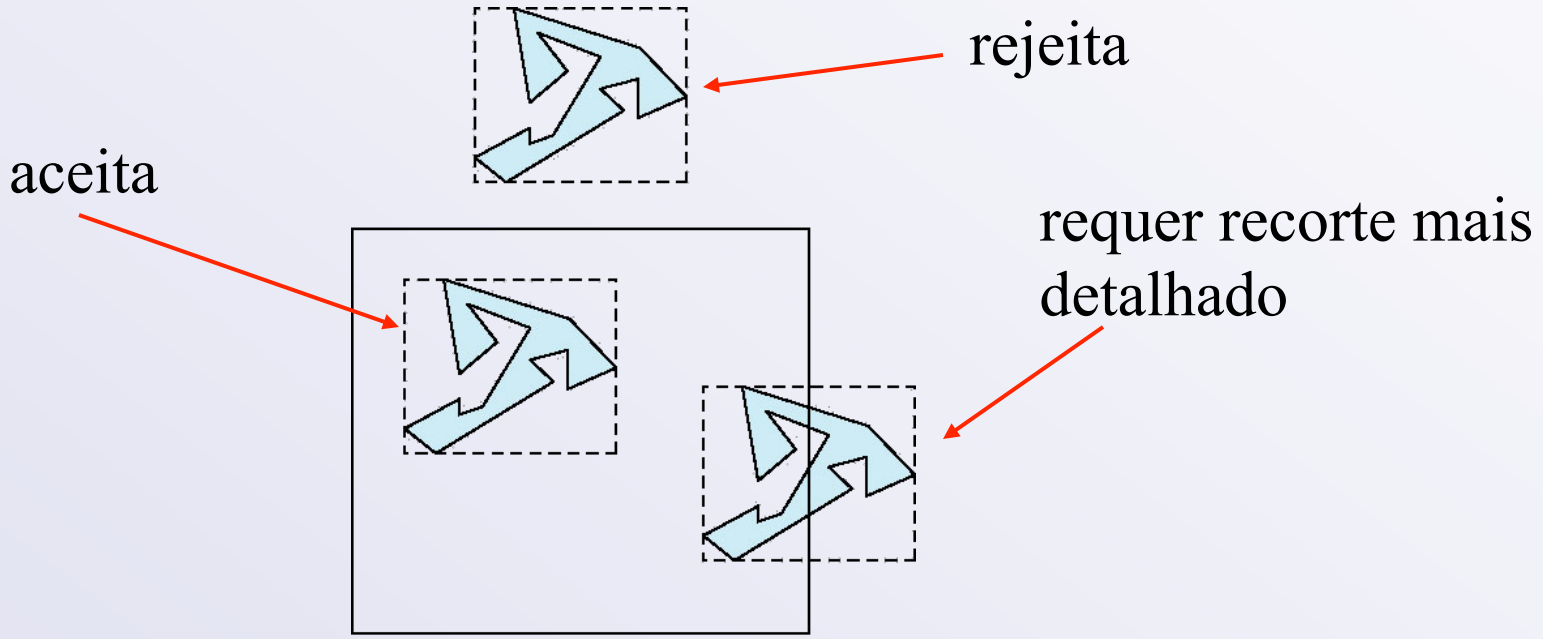
Bounding boxes

- Podemos usar uma caixa de extensão ou “bounding box” para agilizar o processo de recorte de polígonos complexos
  - O menor retângulo alinhado com os eixos ortogonais que engloba um polígono
  - Simples de calcular: max e min de x e y



Bounding boxes

Podemos utilizar essa bounding box para "aceitar" ou rejeitar o polígono



Recorte e Visibilidade

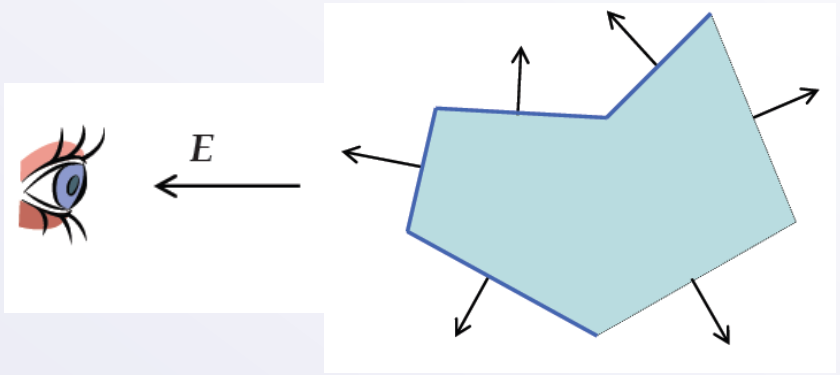
- O processo de recorte têm muito em comum com a remoção de superfícies escondidas;
- Em ambos os casos, estamos tentando remover objetos que não são visíveis ao observador;
- Na maioria das vezes, podemos usar testes de visibilidade para eliminar tantos quantos polígonos possíveis antes de processá-los no pipeline.

Algoritmos de visibilidade

- Visibilidade é um problema complexo que não tem uma solução “ótima”
  - O que é ótima?
    - Pintar apenas as superfícies visíveis?
    - Pintar a cena em tempo mínimo?
  - Coerência no tempo?
    - Cena muda?
    - Objetos se movem?
  - Qualidade é importante?
    - Antialiasing
  - Aceleração por hardware?

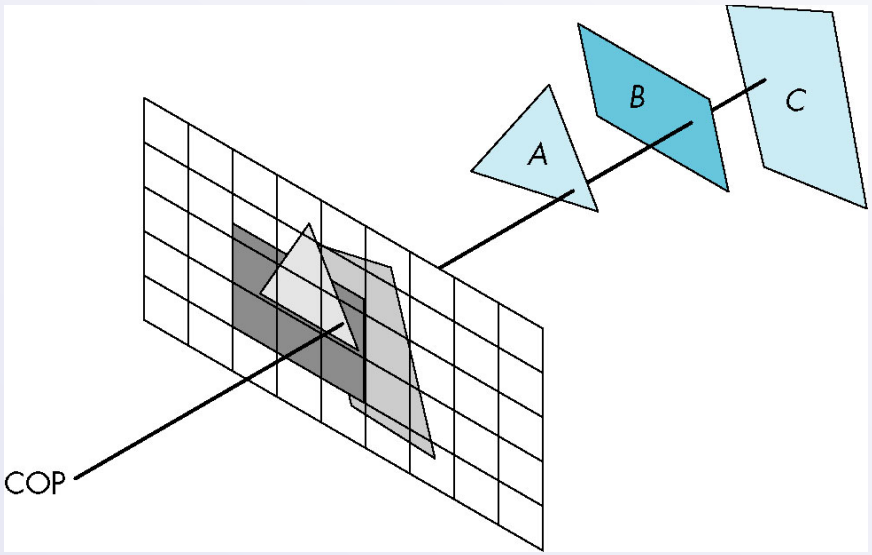
Remoção de Faces Traseiras

- Hipótese: cena é composta de objetos poliédricos fechados
- Podemos reduzir o número de faces aproximadamente à metade
  - Faces de trás não precisam ser pintadas
- Como determinar se a face é de trás?
  - $N \cdot E > 0 \rightarrow$ Face da frente
  - $N \cdot E < 0 \rightarrow$ Face de trás
- Em OpenGL
  - `glEnable(GL_CULLING);`
  - `glCullFace(GL_BACK);`



Espaço de imagem

- Traçar cada raio projetor (nm para um buffer de tamanho $n \times m$) e ache o mais perto dos k polígonos
  - Complexidade $O(nmk)$
  - Ray tracing
- Z-Buffer



Z-Buffer

- Manter para cada pixel um valor de profundidade (*z-buffer* ou *depth buffer*)
- Início da renderização
  - *Buffer* de cor = cor de fundo
  - *z-buffer* = profundidade máxima
- Durante a rasterização de cada polígono, cada pixel passa por um *teste de profundidade*
  - Se a profundidade do pixel for menor que a registrada no *z-buffer*
    - Pintar o pixel (atualizar o buffer de cor)
    - Atualizar o buffer de profundidade
  - Caso contrário, ignorar

Z-Buffer

- OpenGL:
  - Habilitar o z-buffer:

    glEnable (GL_DEPTH_TEST);
  - Não esquecer de alocar o z-buffer
    - glutInitDisplayMode (GLUT_RGB | GLUT_DEPTH);
    - Número de bits por pixel depende de implementação / disponibilidade de memória
  - Ao gerar um novo quadro, limpar também o z-buffer:

    glClear(GL_COLOR_BUFFER_BIT| GL_DEPTH_BUFFER_BIT)
  - Ordem imposta pelo teste de profundidade pode ser alterada
    - Ex: glDepthFunc (GL_GREATER);

Z-Buffer

- Vantagens:
  - Simples e comumente implementado em Hardware
  - Objetos podem ser desenhados em qualquer ordem
- Desvantagens:
  - Rasterização independe de visibilidade
    - Lento se o número de polígonos é grande
  - Erros na quantização de valores de profundidade podem resultar em imagens inaceitáveis
  - Dificulta o uso de transparência ou técnicas de anti-serrilhado
    - É preciso ter informações sobre os vários polígonos que cobrem cada pixel

Algoritmo de Recorte Sucessivo

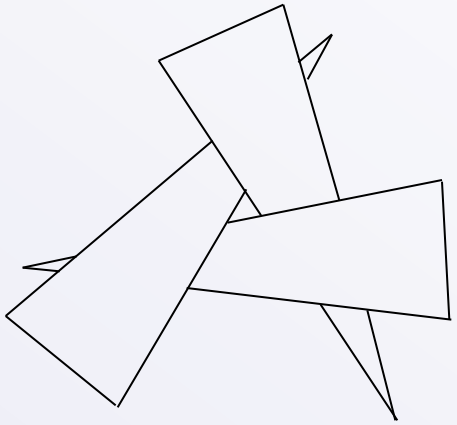
- Pode ser pensado como um algoritmo do pintor ao contrário
- Polígonos são pintados de frente para trás
- É mantida uma máscara que delimita quais porções do plano já foram pintadas
  - Máscara é um polígono genérico (pode ter diversas componentes conexas e vários “buracos”)
- Ao considerar cada um novo polígono *P*
  - Recortar contra a máscara *M* e pintar apenas *P* – *M*
  - Máscara agora é *M* + *P*

Algoritmo de Recorte Sucessivo

- Vantagens
  - Trabalha no espaço do objeto
    - Independe da resolução da imagem
    - Não tem problemas de quantização em *z*
  - Pinta cada pixel uma vez apenas
- Desvantagem
  - Máscara pode se tornar arbitrariamente complexa
  - Excessivamente lento

Algoritmo do Pintor

- Também conhecido como algoritmo de prioridade em *Z* (*depth priority*)
- Idéia é pintar objetos mais distantes (*background*) antes de pintar objetos próximos (*foreground*)
- Requer que objetos sejam ordenados em *Z*
  - Complexidade O (*N* log *N*)
  - Pode ser complicado em alguns casos
  - Na verdade, a ordem não precisa ser total se projeções dos objetos não se interceptam

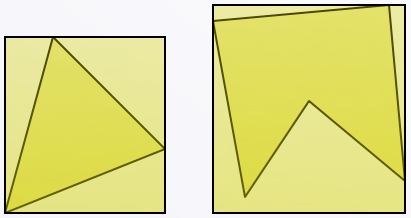


Não há ordem possível

Algoritmo do Pintor

- Ordenação requer que se determine, para cada par de polígonos *A* e *B:*
  - *A* precisa ser pintado antes de *B*
  - *B* precisa ser pintado antes de *A*
  - A ordem de pintura é irrelevante
- Pode-se usar um algoritmo simples baseado em troca.
  - Ex.: *Bubble Sort*

Algoritmo do Pintor

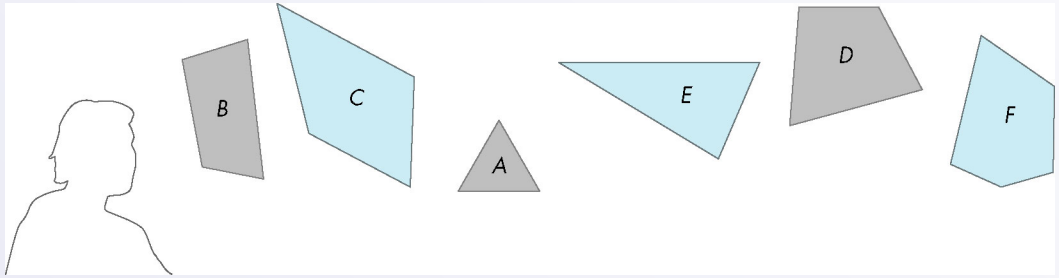


- Ordem de pintura entre *A* e *B* determinada por testes com níveis crescentes de complexidade
  1. Caixas limitantes de *A* e *B* não se interceptam
  2. *A* atrás ou na frente do plano de *B*
  3. *B* atrás ou na frente do plano de *A*
  4. Projeções de *A* e *B* não se interceptam
- Se nenhum teste for conclusivo, A é substituído pelas partes obtidas recortando A pelo plano de B (ou vice-versa)

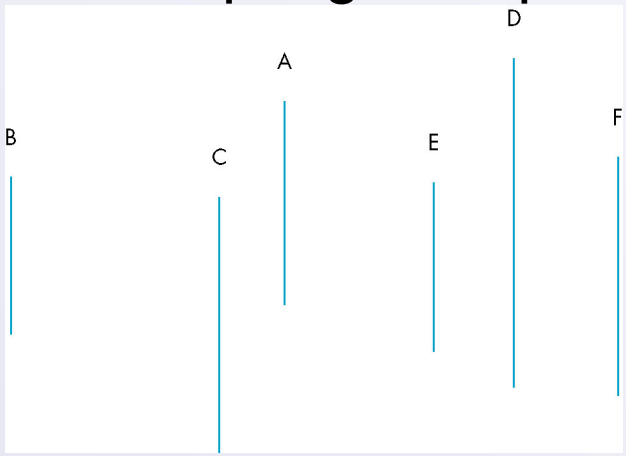
Árvores BSP

- São estruturas de dados que representam uma partição recursiva do espaço
- Muitas aplicações em computação gráfica
- Estrutura multi-dimensional
- Cada célula (começando com o espaço inteiro) é dividida em duas por um plano
  - *Binary Space Partition Tree*
- Partição resultante é composta de células convexas

Exemplo



considere 6 polígonos paralelos

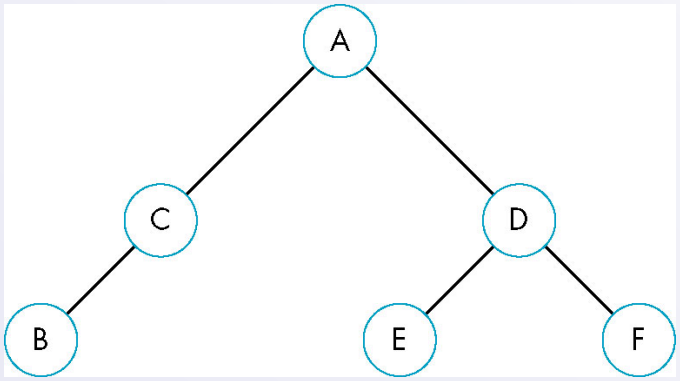


visão superior

O plano de A separa B e C de D, E e F

Árvore BSP

- Podemos continuar recursivamente
  - Plano de C separa B de A
  - Plano de D separa E e F
- Esta informação pode ser inserida em uma árvore BSP
  - Teste de visibilidade



Árvores BSP

- Vantagens
  - Pode ser usado para caminhadas
  - Filtragem e anti-aliasing suportados com facilidade (desenho de trás para a frente)
- Desvantagens
  - Objetos em movimento
  - Desenha mesmo pixel várias vezes
  - Número de polígonos pode crescer muito

Tarefa de casa

- Como árvores BSP podem ser usadas para mapear ambientes dinâmicos (com objetos em movimento) ?
  - Vantagens ?
- Começe o EP #2!
  - Inicie a modelagem do mundo IME virtual