Computação Gráfica

Ana Paula Piovesan Melchiori

Recorte

Algoritmos

Recorte (*Clipping*)

- Problema definido por
  - Geometria a ser recortada
    - Pontos, retas, planos, curvas, superfícies
  - Restrições de recorte
    - Janela (2D)
    - Volume de visibilidade
      - Frustum (tronco de pirâmide)
      - Paralelepípedo
    - Polígonos
      - Convexos
      - Genéricos (côncavos, com buracos, etc)
- Resultado depende da geometria
  - Pontos: valor booleano (visível / não visível)
  - Retas: segmento de reta ou coleção de segmentos de reta
  - Planos: polígono ou coleção de polígonos

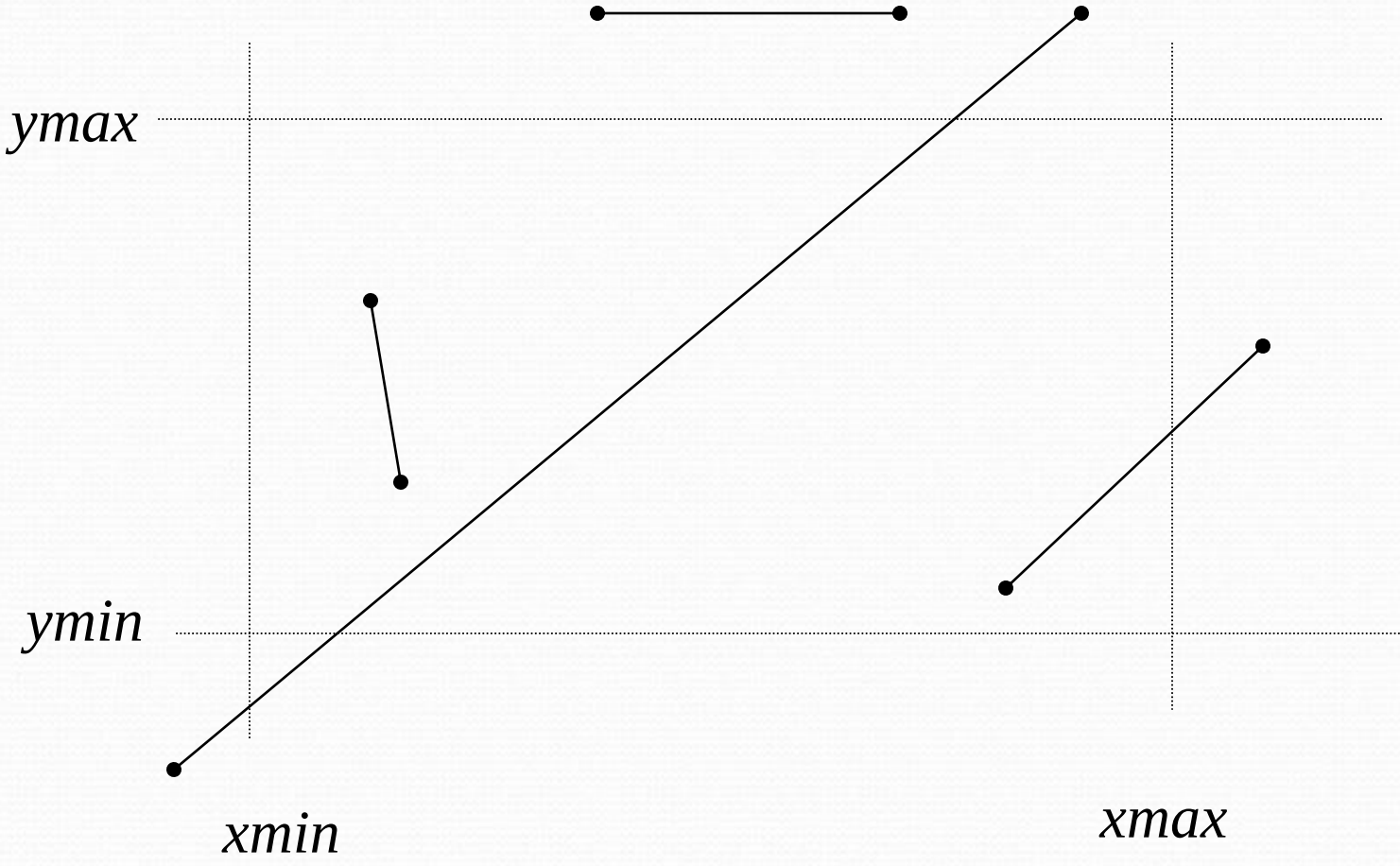
Recorte de Segmento de Reta x Retângulo

- Problema clássico 2D
- Entrada:
  - Segmento de reta P_1 - P_2
  - Janela alinhada com eixos (*xmin*, *ymin*) - (*xmax*, *ymax*)
- Saída: Segmento recortado (possivelmente nulo)
- Variantes
  - Cohen-Sutherland
  - Liang-Barksy / Cyrus-Beck
  - Nicholl-Lee-Nicholl

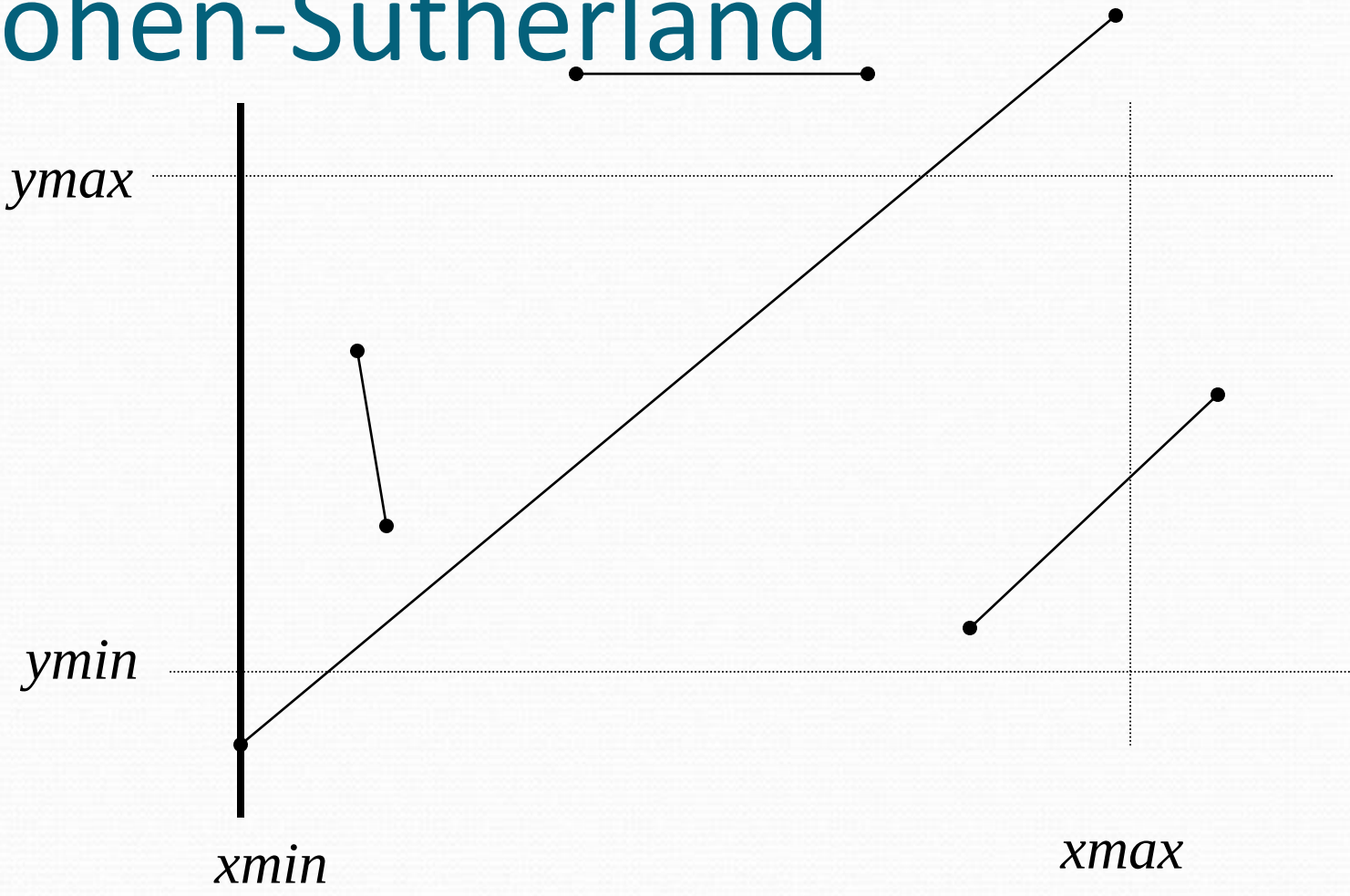
Cohen-Sutherland

- Vértices do segmento são classificados com relação a cada semi-espaço plano que delimita a janela
  - $x \geq xmin$ e $x \leq xmax$ e $y \geq ymin$ e $y \leq ymax$
- Se ambos os vértices classificados como fora, descartar o segmento (totalmente invisível)
- Se ambos classificados como dentro, testar o próximo semi-espaço
- Se um vértice dentro e outro fora, computar o ponto de interseção Q e continuar o algoritmo com o segmento recortado (P_1-Q ou P_2-Q)

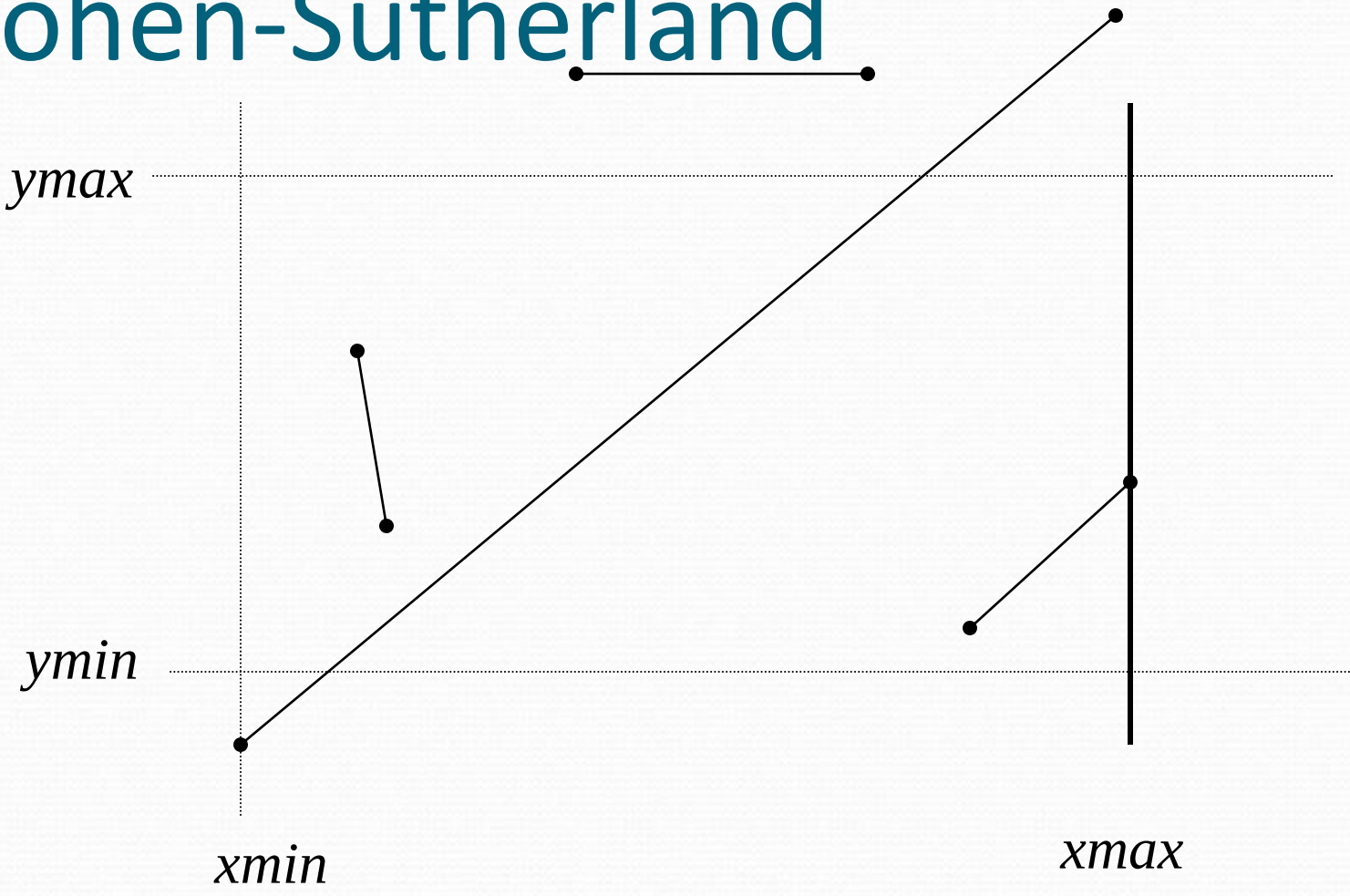
Cohen-Sutherland



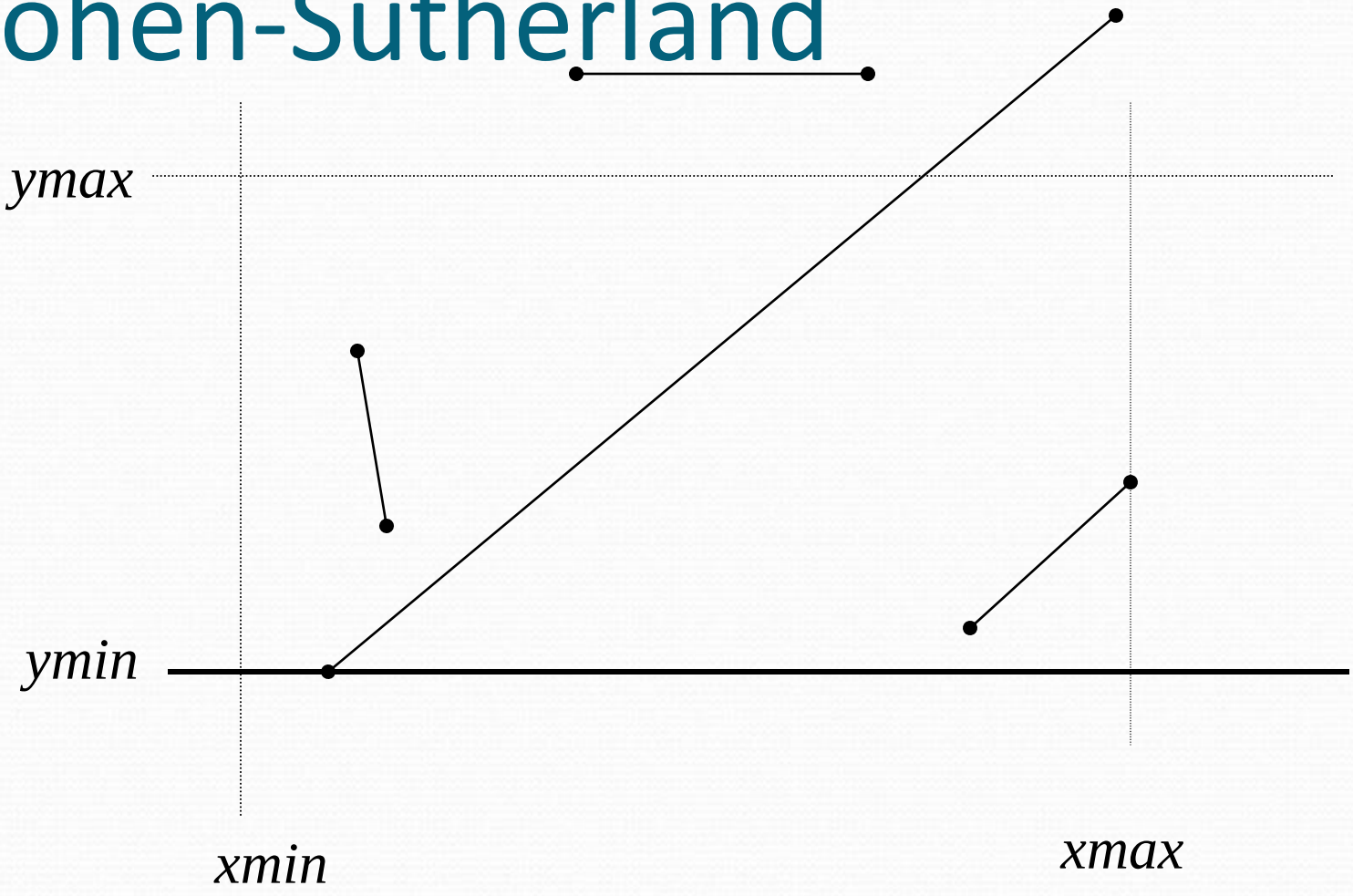
Cohen-Sutherland



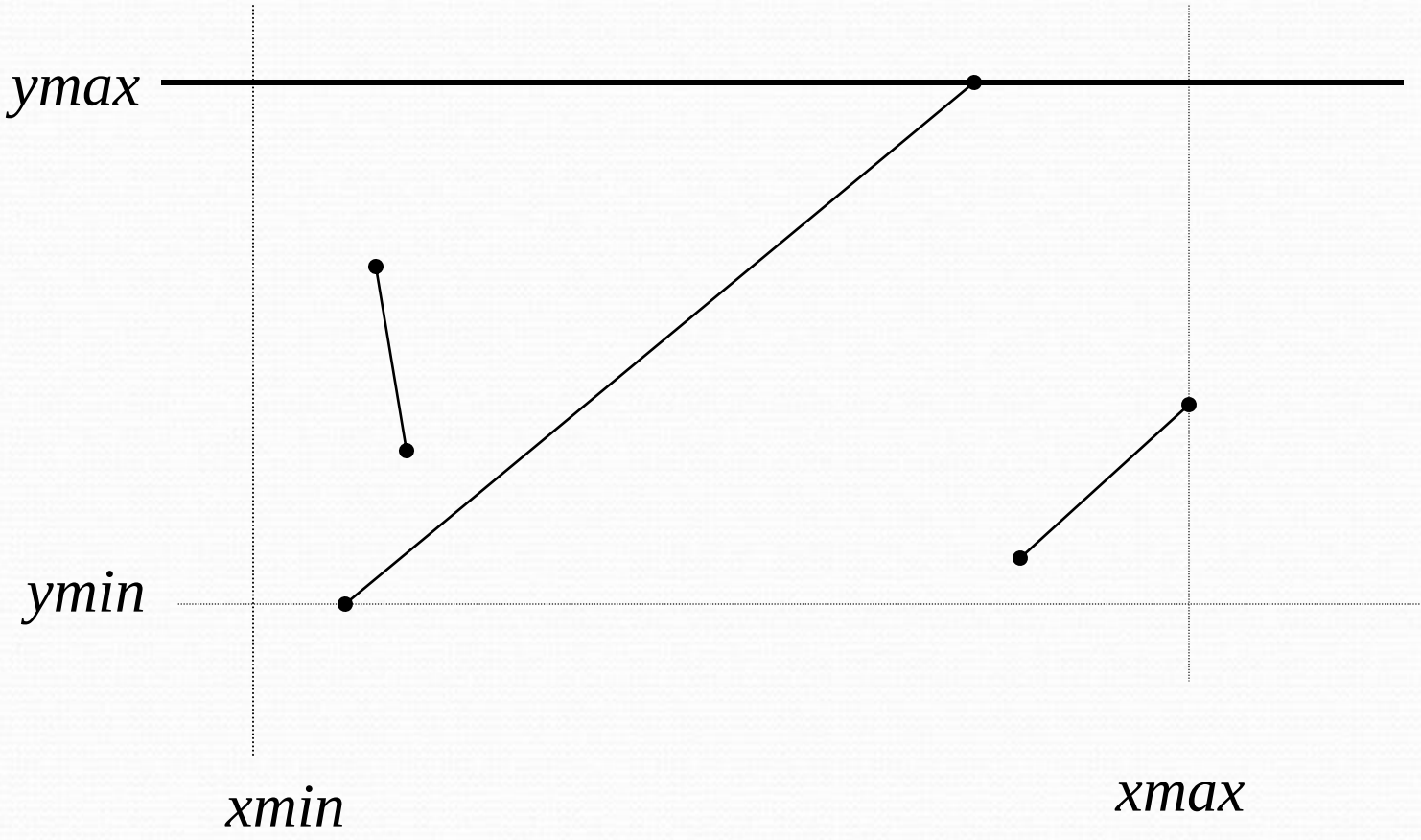
Cohen-Sutherland



Cohen-Sutherland

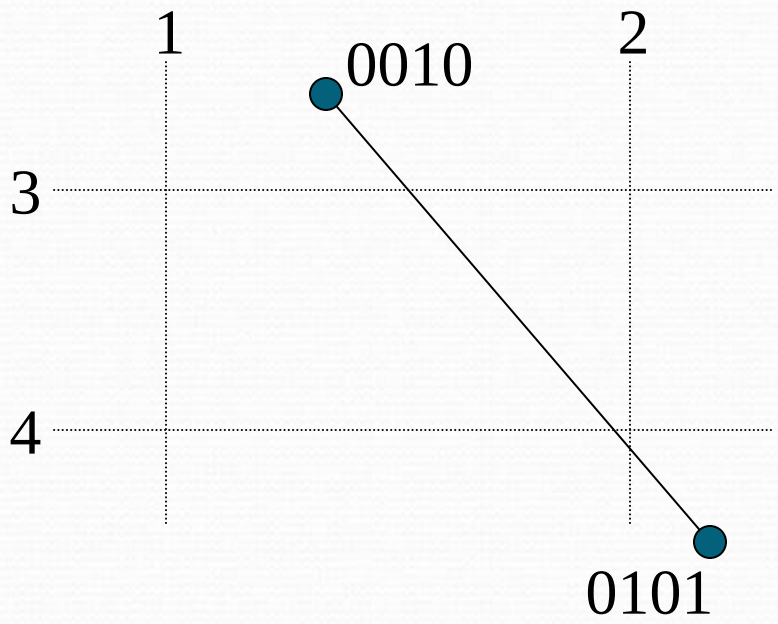


Cohen-Sutherland



Cohen-Sutherland - Detalhes

- Recorte só é necessário se um vértice dentro e outro fora
- Classificação de cada vértice pode ser codificada em 4 bits, um para cada semi-espaço
  - Dentro = 0 e Fora = 1
- Rejeição trivial:
  - Classif(P_1) & Classif(P_2) ≠ 0
- Aceitação trivial:
  - Classif(P_1) | Classif(P_2) = 0
- Interseção com quais semi-espaços?
  - Classif(P_1) ^ Classif(P_2)



Algoritmo de Liang-Barsky

- Refinamento que consiste em representar a reta em forma paramétrica
- É mais eficiente visto que não precisamos computar pontos de interseção irrelevantes
- Porção da reta não recortada deve satisfazer

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

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

Algoritmo de Liang-Barsky

- Linha infinita intercepta semi-espaços planos para os seguintes valores do parâmetro t:

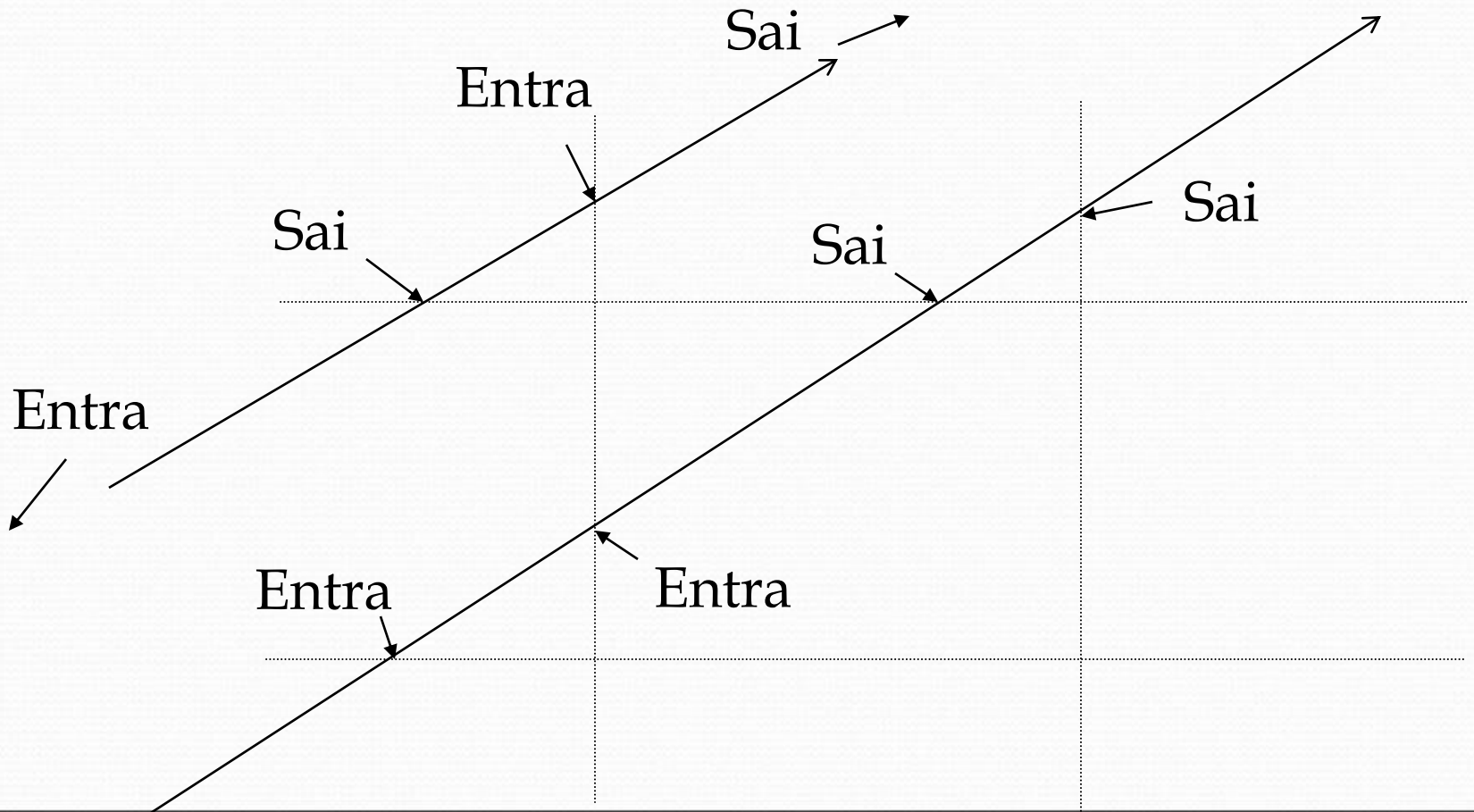
$$t_k = \frac{q_k}{p_k} \qquad \text{onde}$$

$$p_1 = -\Delta x \quad q_1 = x_1 - x_{\min}$$
$$p_2 = \Delta x \quad q_2 = x_{\max} - x_1$$
$$p_3 = -\Delta y \quad q_3 = y_1 - y_{\min}$$
$$p_4 = \Delta y \quad q_4 = y_{\max} - y_1$$

Algoritmo de Liang-Barsky

- Se $p_k < 0$, à medida que t aumenta, reta **entra** no semi-espaço plano
- Se $p_k > 0$, à medida que t aumenta, reta **sai** do semi-espaço plano
- Se $p_k = 0$, reta é paralela ao semi-espaço plano (recorte é trivial)
- Se existe um segmento da reta dentro do retângulo, classificação dos pontos de interseção deve ser **entra, entra, sai, sai**

Algoritmo de Liang-Barsky



Liang-Barsky – Pseudo-código

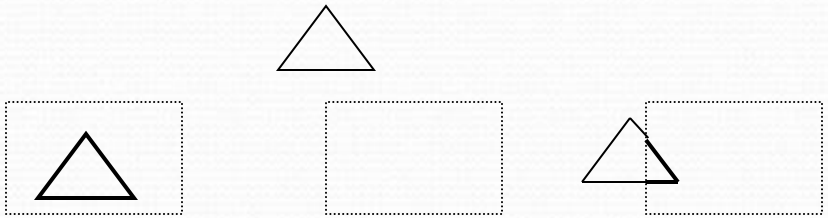
- Computar valores de *t* para os pontos de interseção
- Classificar pontos em **entra** ou **sai**
- Vértices do segmento recortado devem corresponder a dois valores de *t:*
  - t_{min}= max (0, *t*'s do tipo **entra**)
  - t_{max}= min (1, *t*'s do tipo **sai**)
- Se $t_{min}<t_{max}$, segmento recortado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

Recorte de Polígono contra Retângulo

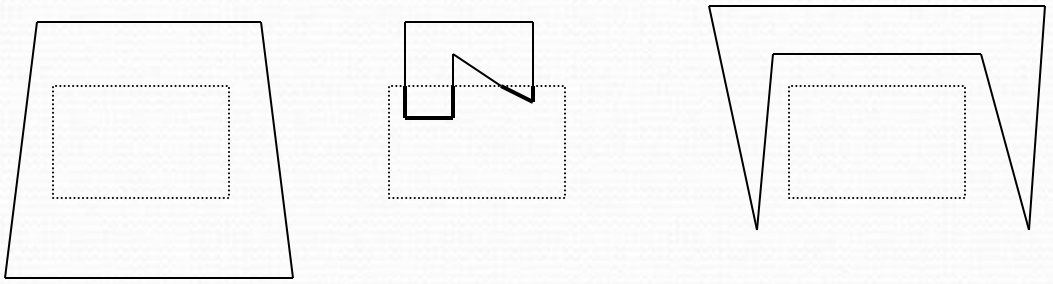
- Inclui o problema de recorte de segmentos de reta
  - Polígono resultante tem vértices que são
    - Vértices da janela,
    - Vértices do polígono original, ou
    - Pontos de interseção aresta do polígono/aresta da janela
- Dois algoritmos clássicos
  - Sutherland-Hodgman
    - Figura de recorte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

Recorte de Polígono contra Retângulo

- Casos Simples

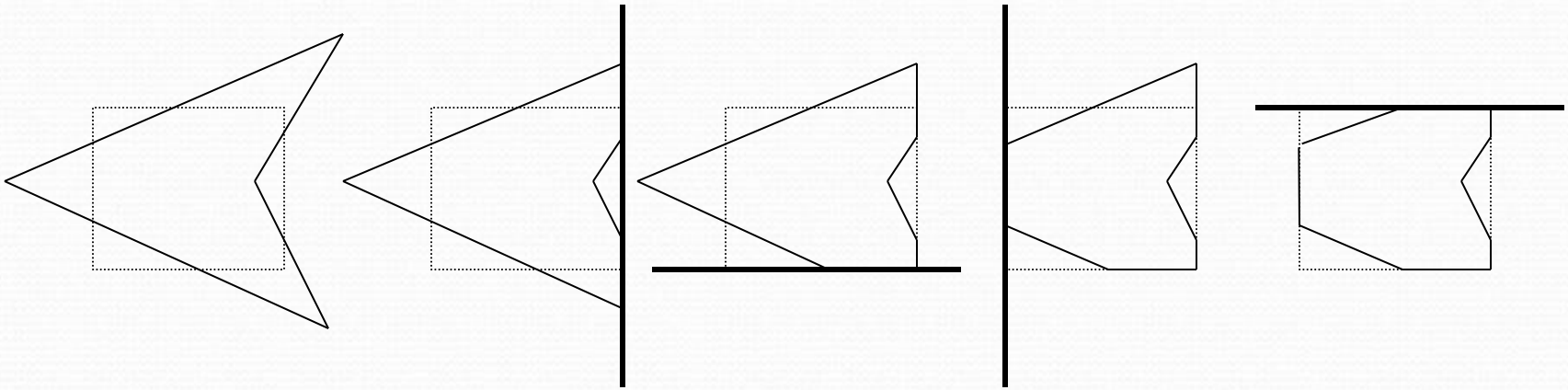


- Casos Complicados



Algoritmo de Sutherland-Hodgman

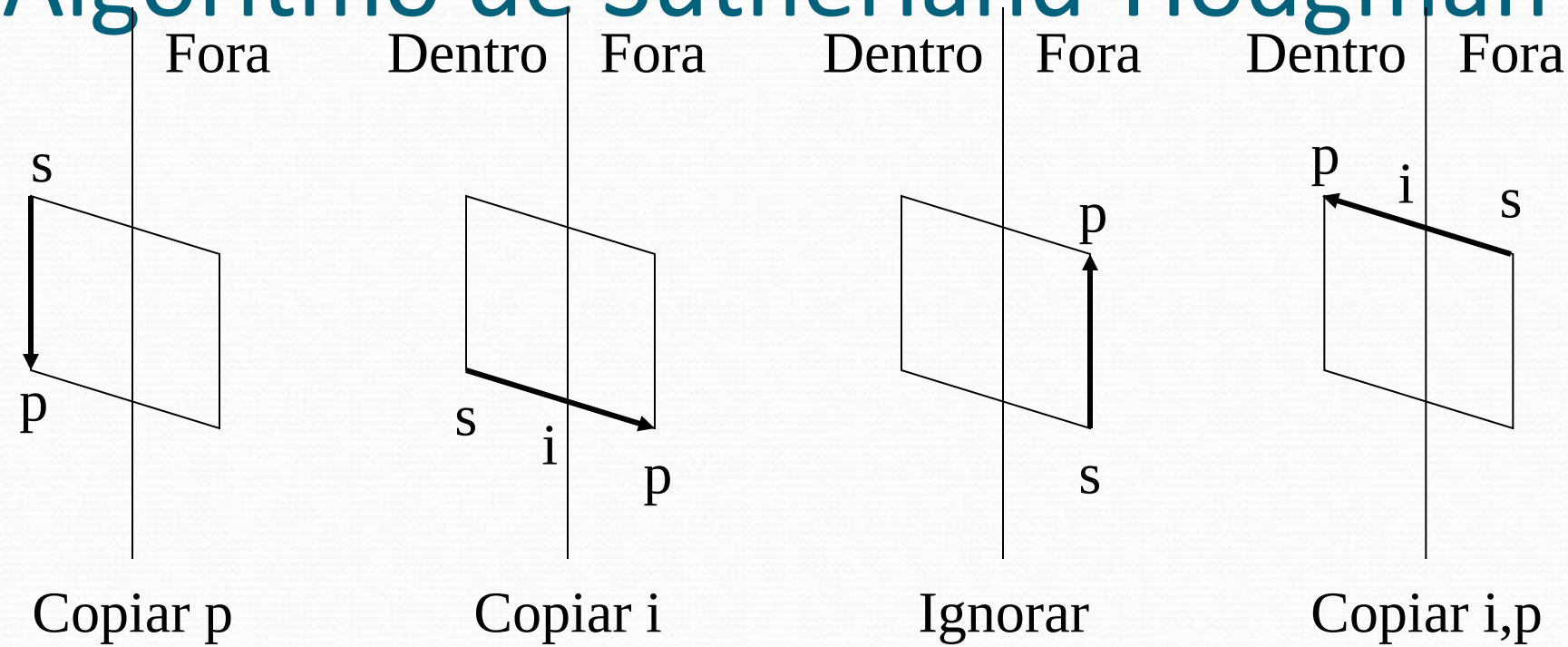
- Idéia é semelhante à do algoritmo de Sutherland-Cohen
  - Recortar o polígono sucessivamente contra todos os semi-espaços planos da figura de recorte



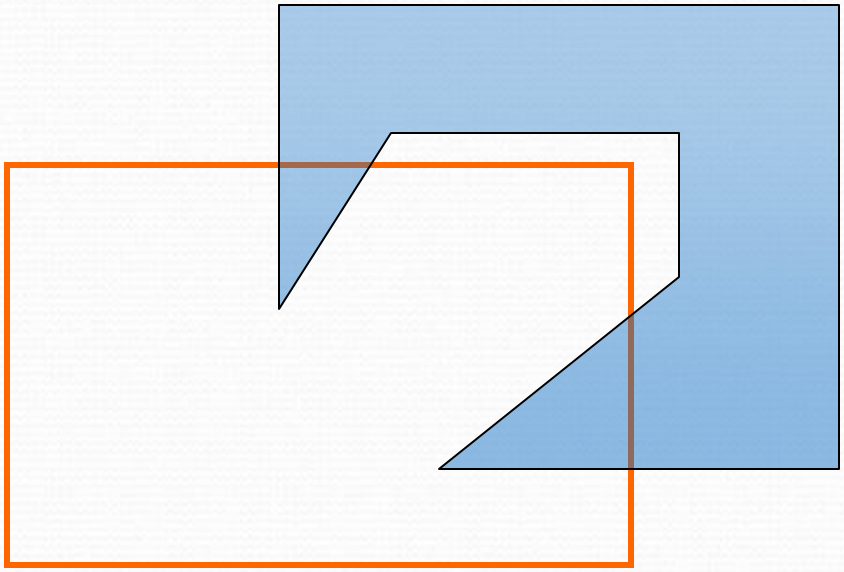
Algoritmo de Sutherland-Hodgman

- Polígono é dado como uma lista circular de vértices
- Vértices e arestas são processados em seqüência e classificados contra o semi-espaço plano corrente
  - Vértice:
    - Dentro: copiar para a saída
    - Fora: ignorar
  - Aresta
    - Intercepta semi-espaço plano (vértice anterior e posterior têm classificações diferentes) : Copiar ponto de interseção para a saída
    - Não intercepta: ignorar

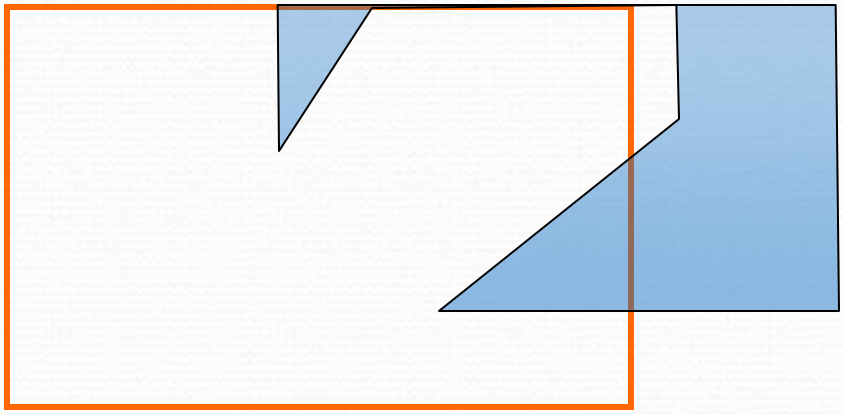
Algoritmo de Sutherland-Hodgman



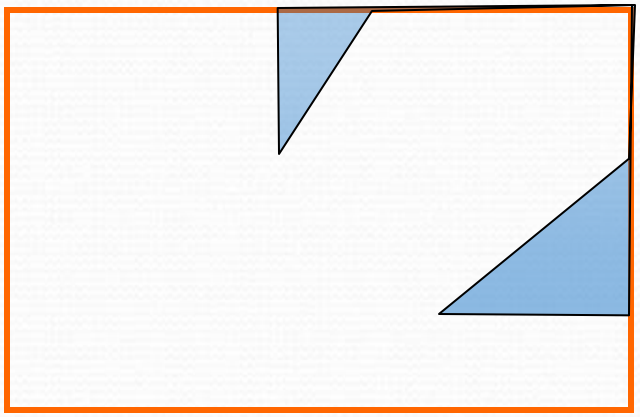
Sutherland-Hodgman – Exemplo



Sutherland-Hodgman – Exemplo

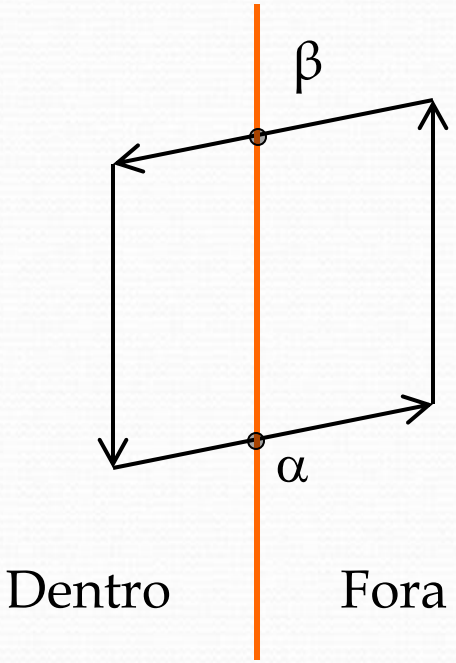


Sutherland-Hodgman – Exemplo

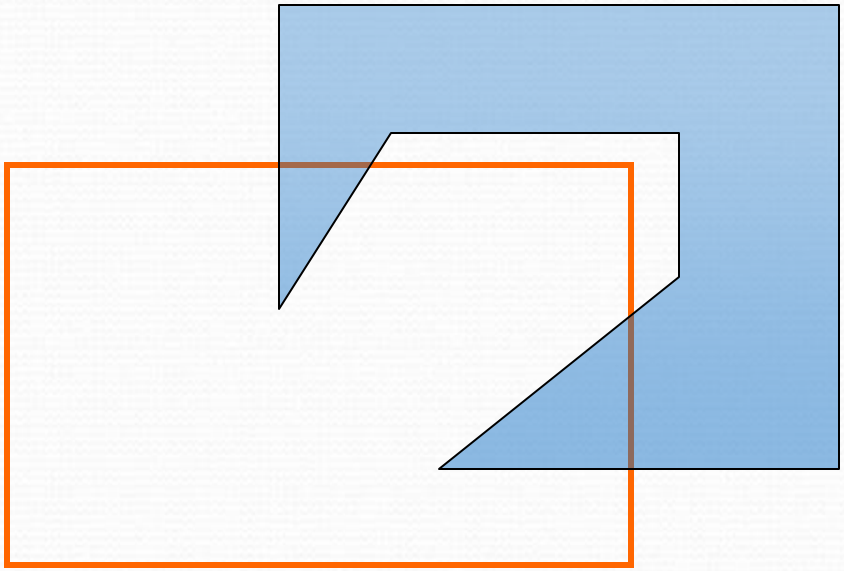


Sutherland Hodgman – Eliminando Arestas Fantasmas

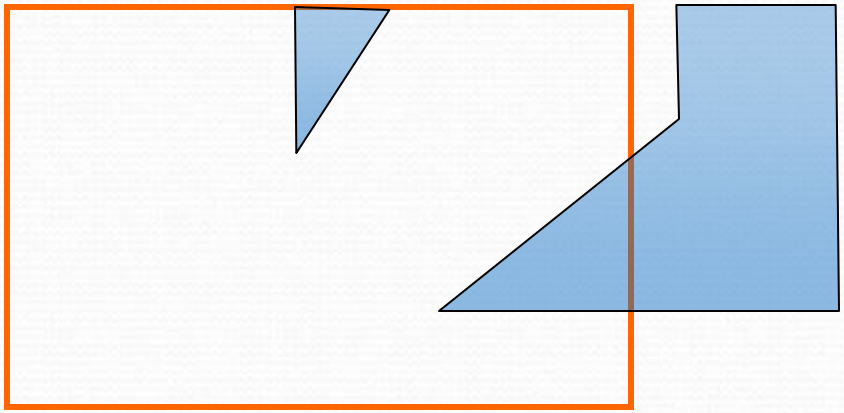
- Distinguir os pontos de interseção gerados
  - De dentro para fora: rotular como do tipo α
  - De fora para dentro: rotular como do tipo β
- Iniciar o percurso de algum vértice "fora"
- Ao encontrar um ponto de interseção α, ligar com o último β visto
- Resultado pode ter mais de uma componente conexa



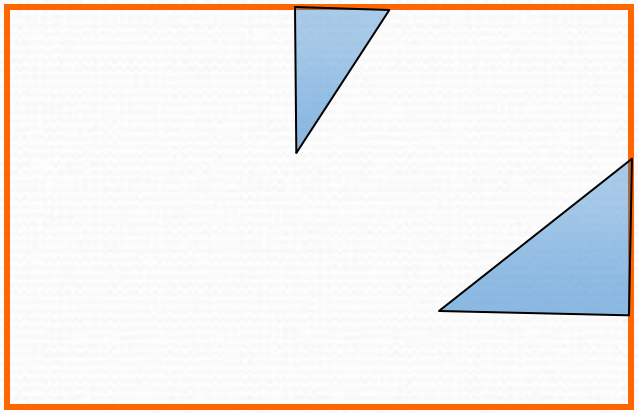
Sutherland Hodgman – Eliminando Arestas Fantasmas – Exemplo



Sutherland Hodgman – Eliminando Arestas Fantasmas – Exemplo



Sutherland Hodgman – Eliminando Arestas Fantasmas – Exemplo



Sutherland-Hodgman - Resumo

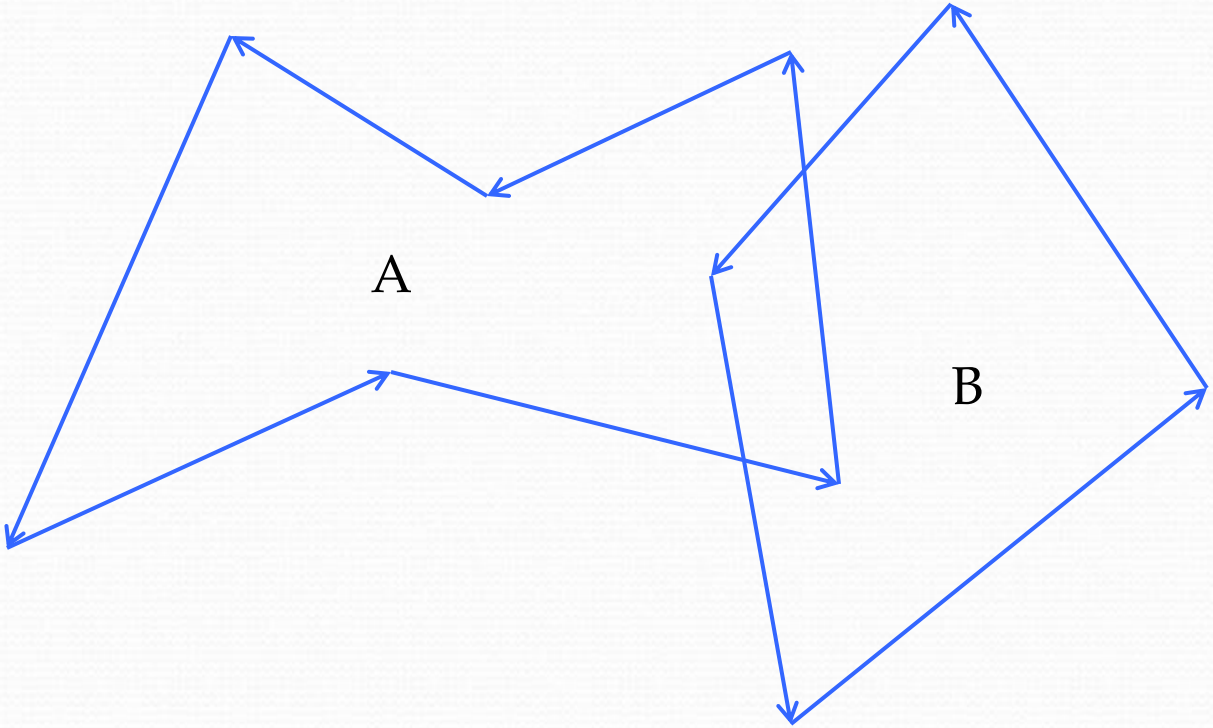
- Facilmente generalizável para 3D
- Pode ser adaptado para implementação em hardware
  - Cada vértice gerado pode ser passado pelo pipeline para o recorte contra o próximo semi-espaço plano
- Pode gerar arestas “fantasma”
  - Irrelevante para propósitos de desenho
  - Podem ser eliminadas com um pouco mais de trabalho

Algoritmo de Weiler-Atherton

- Recorta qualquer polígono contra qualquer outro polígono
- Pode ser usado para computar operações de conjunto com polígonos
  - União, Interseção, Diferença
- Mais complexo que o algoritmo de Sutherland-Hodgman
- Idéia:
  - Cada polígono divide o espaço em 3 conjuntos
    - Dentro, fora, borda
  - Borda de cada polígono é “duplicada”
    - Uma circulação corresponde ao lado de dentro e outra ao lado de fora
  - Nos pontos de interseção, é preciso “costurar” as 4 circulações de forma coerente

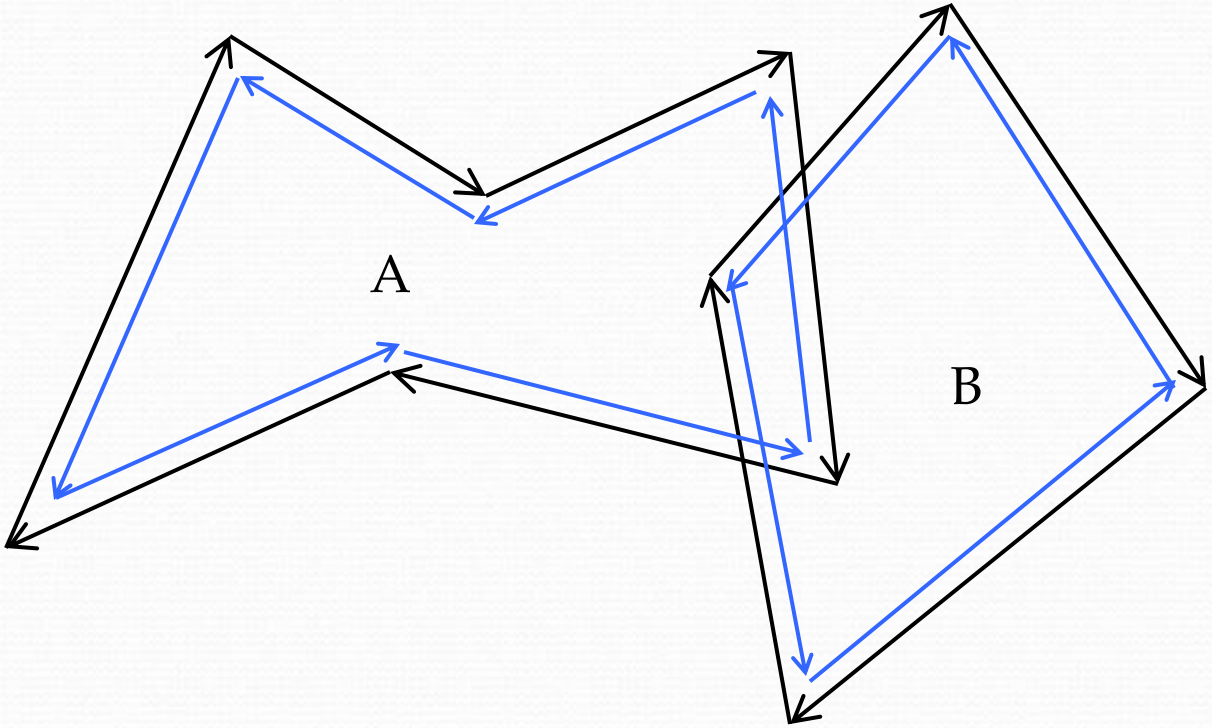
Algoritmo de Weiler-Atherton

Interior do polígono à esquerda da seta
(circulação anti-horária)



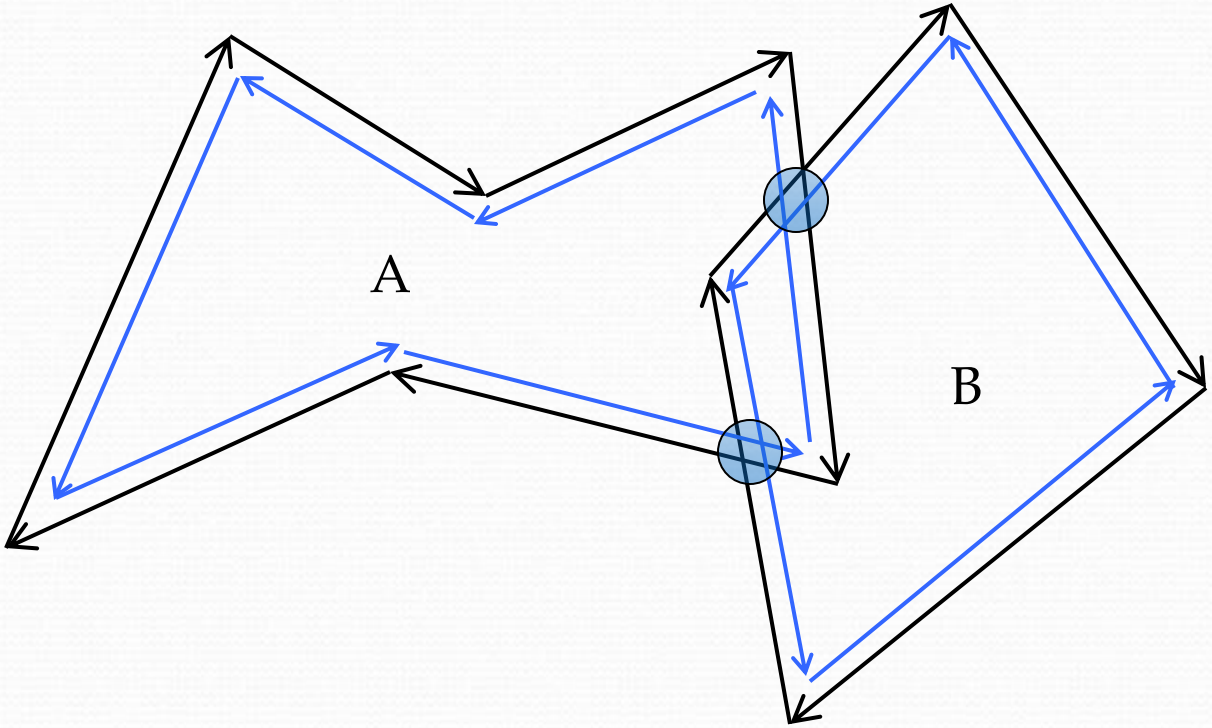
Algoritmo de Weiler-Atherton

Exterior do polígono à direita da seta
(circulação horária)



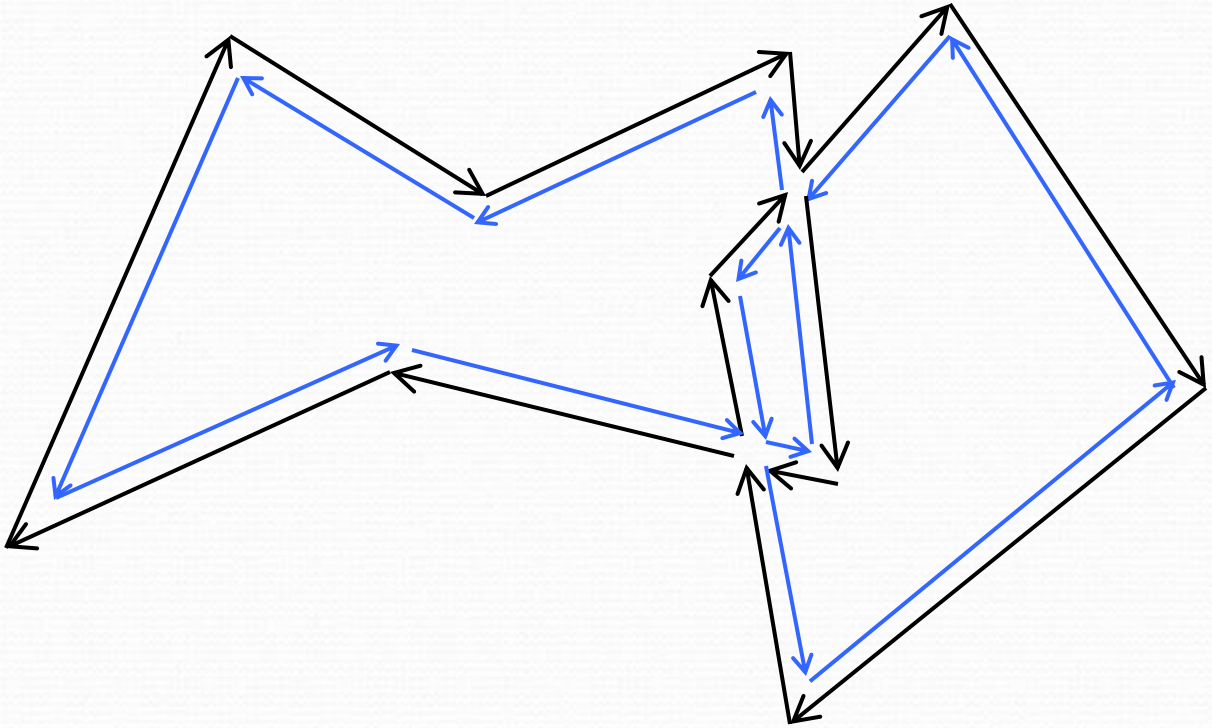
Algoritmo de Weiler-Atherton

Pontos de interseção são calculados

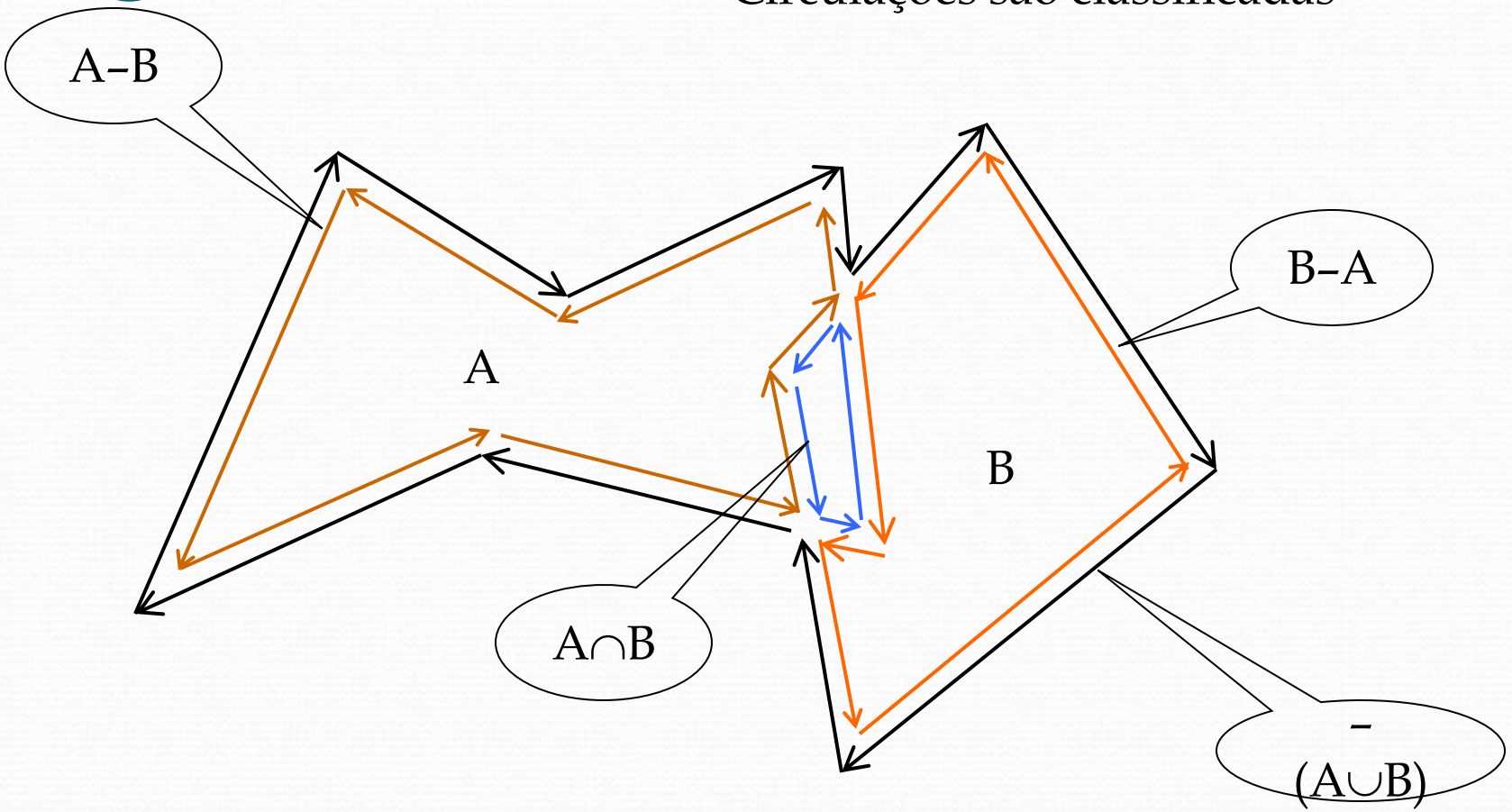


Algoritmo de Weiler-Atherton

Circulações são costuradas



Algoritmo de Weiler-Atherton



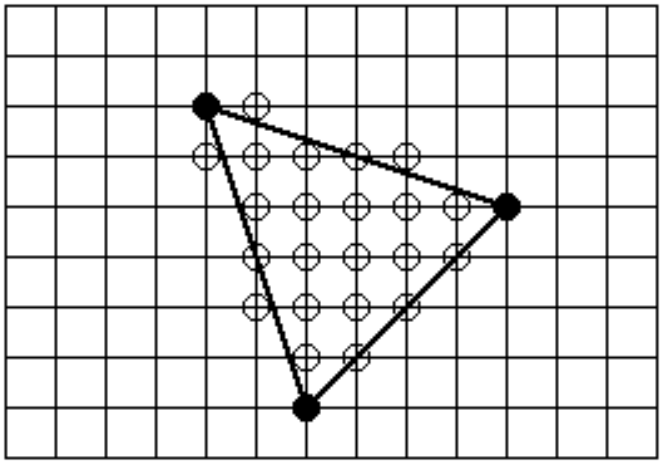
Algoritmos de Colorização

Colorização

- É também chamado de shading, e está relacionado com a interpolação de cores, ou seja, é a aplicação dos modelos de iluminação. É a função, chamada função de intensidade de cor Ic, definida no espaço da tela virtual e dada por
- $Ic(x,y) = I(x',y',z')$
- onde $I(x',y',z')$ é o ponto visível da cena que corresponde ao ponto (x,y) da tela

Colorização

- Só se tem o valor de Ipixel nos vértices, que são armazenados na lista e que são colocados no pipeline gráfico
- Algoritmos de rasterização e preenchimento de área.E o valor de cor nos outros pixels ?

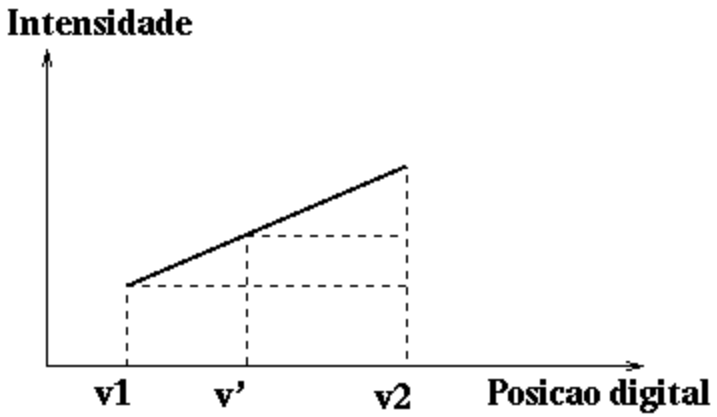


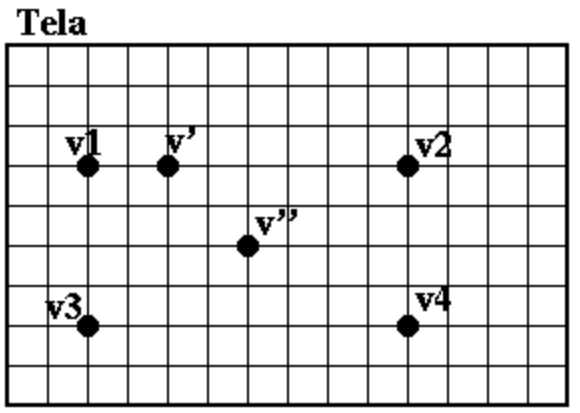
Colorização

- O domínio de Ic é a tela virtual, que é um reticulado uniforme de pontos
- A região digital da tela virtual (que é um subconjunto, isto é, um conjunto finito de pixels) determinada no processo de rasterização é o domínio da função de colorização
- Deste modo deve-se calcular o valor da função Ic em cada pixel desta região
- Este processo de amostragem é uma das etapas mais delicadas no cálculo da função Ic, pois não deve haver perda de informação neste processo
  - Aliasing

Colorização

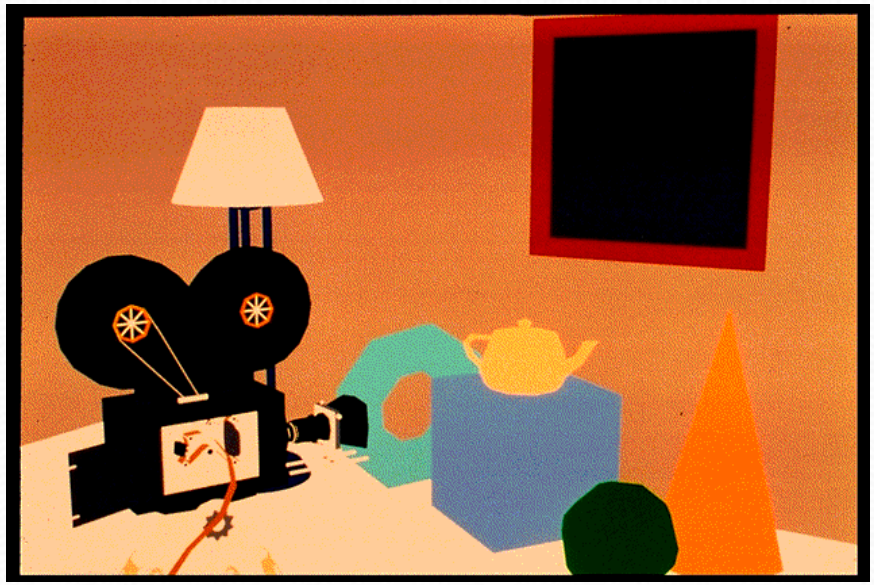
- Entre v1 e v2 tem-se a intensidade em v'
- Ic(v') = Ic(v1) + Dv [Ic(v2) - Ic(v1)]
- onde Dv varia de 1 até n - 1, ou seja, interpolação linear
- Para se achar o valor em v'', tem-se uma interpolação bilinear





Colorização

- Uma função de colorização primitiva pode ser o algoritmo de preenchimento de regiões
- Iluminação é somente ambiente, ou seja, iluminação uniforme (constante) de todas as direções
  - I = ka Ia
  - A desvantagem é que se perde informação 3D porque o objeto é iluminado uniformemente



Colorização

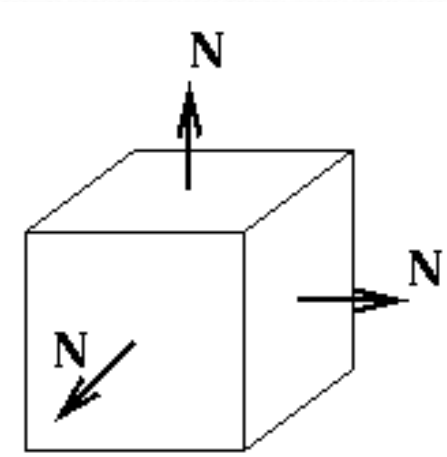
- A interpolação tem por finalidade evitar o cálculo da função de iluminação para todos os pixels.
- Se os objetos são representados por B-rep poligonal, então a região digital visível é formada por um conjunto de polígonos Não serão vistas as técnicas de visualização para representação CSG
  - Neste caso, é possível fazer a conversão por meio de uma poligonalização

Colorização

- Existem 3 métodos para o cálculo de Ic em regiões poligonais, cada um associado a um modelo local de iluminação
  - Colorização constante
  - Colorização de Gouraud
  - Colorização de Phong

Colorização Constante

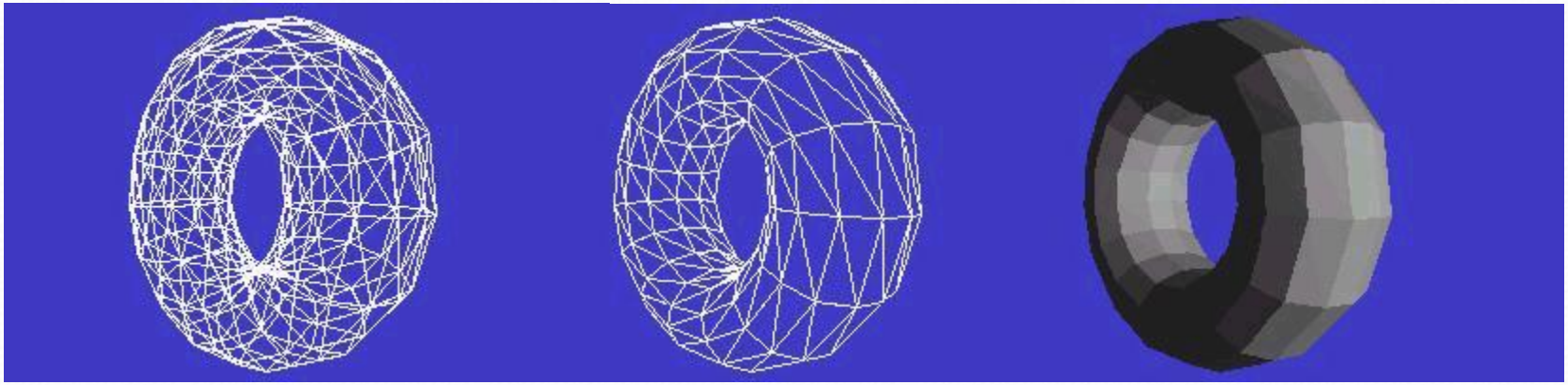
- É também chamada de flat shading ou colorização facetada.
- Como os objetos da cena utilizam uma representação poligonal, significa que há uma normal para cada face.



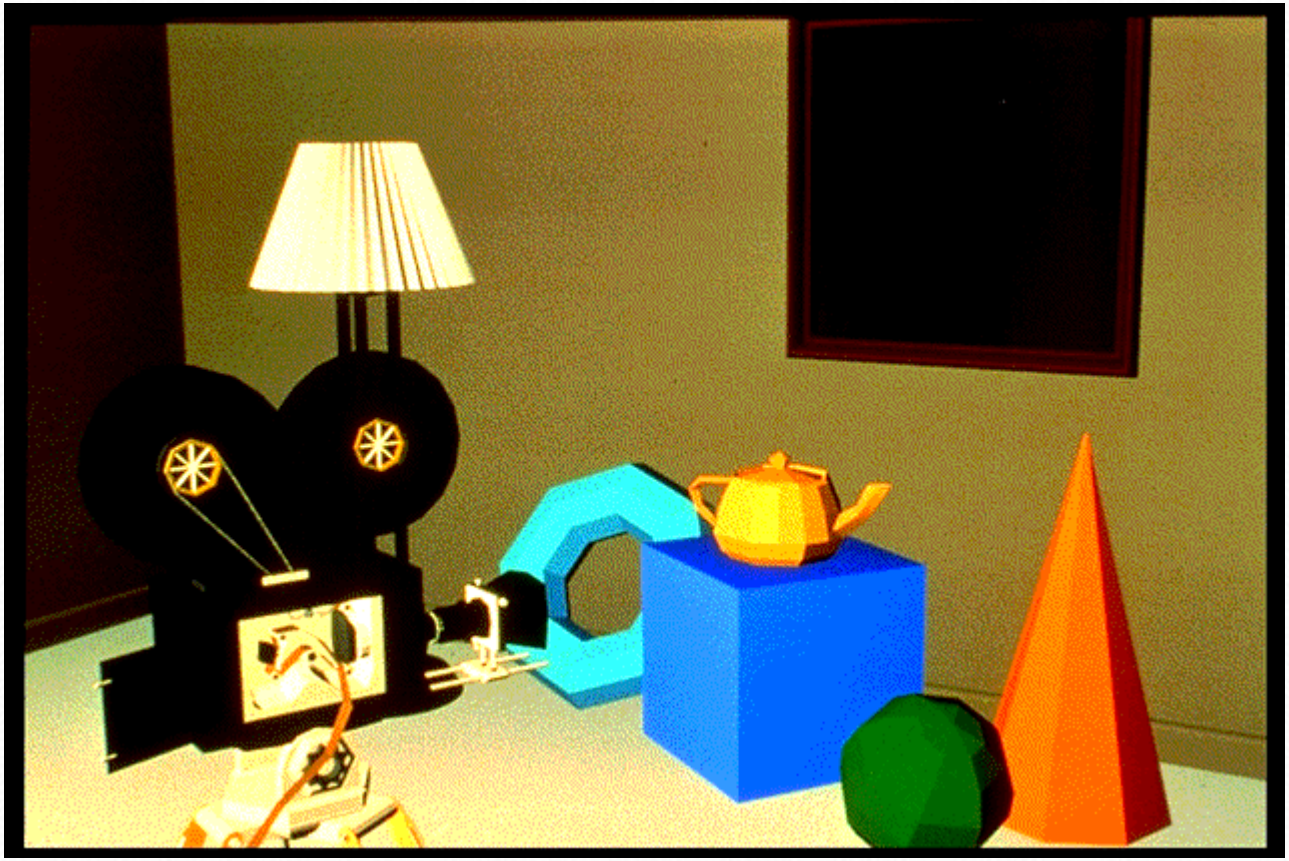
Colorização Constante

- Considerando-se a fonte de luz no infinito, significa que cada face tem a mesma côr para os pixels da tela virtual. Estas são as condições assumidas por Bouknight (1970) na definição deste método de colorização
  - Vantagem: Tem-se uma quantidade mínima de cálculo para a função de iluminação, ou seja, é o método computacionalmente mais eficiente
  - Desvantagem: Tem-se nitidamente a aproximação poligonal utilizada, dada a imagem facetada resultante.

Colorização Constante



Colorização Constante



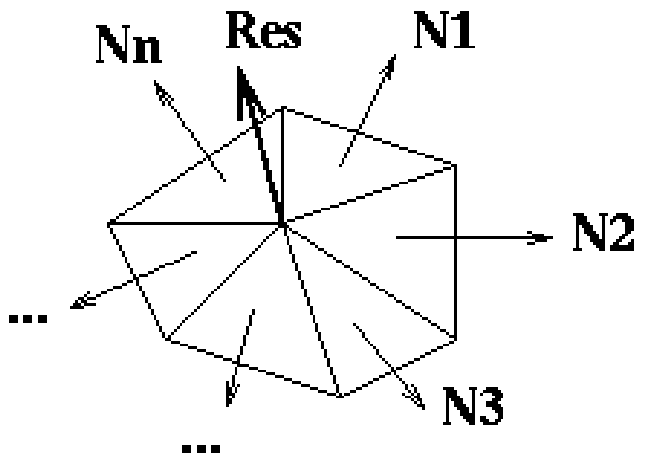
Colorização de Gouraud

- É também chamada de Gouraud shading.
- O modelo B-rep poligonal é apenas uma aproximação do objeto na cena e é linear. Daí, Henri Gouraud (em 1971) usou a interpolação poligonal para calcular o valor da função Ic nos pontos interiores do polígono.
- No método, calcula-se uma normal em cada vértice da B-rep poligonal.

Colorização de Gouraud

- Esta normal aproxima a normal à superfície original.
- Esta normal depende da curvatura da superfície no ponto.
- Se a superfície estiver definida implicitamente, a normal é dada pelo gradiente calculado no vértice.
- Caso contrário, o cálculo da normal é efetuado a partir da média das normais de todas as faces.

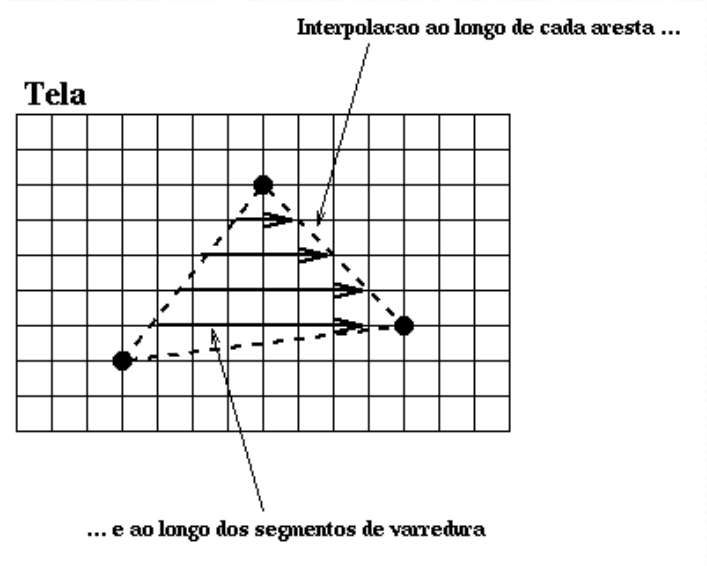
Colorização de Gouraud



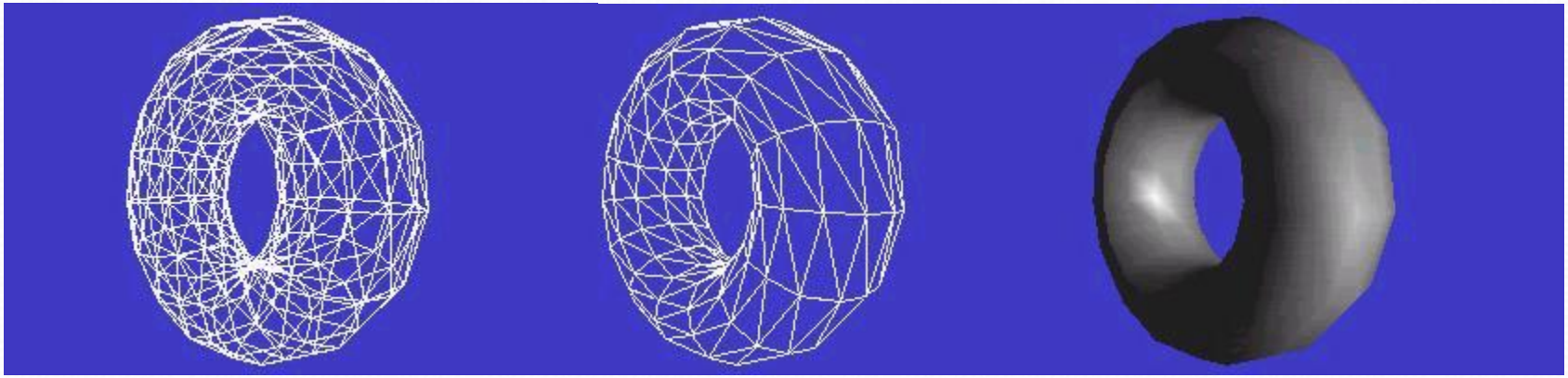
$$Res = \frac{1}{n} \sum_{i=1}^{n} N_i$$

Colorização de Gouraud

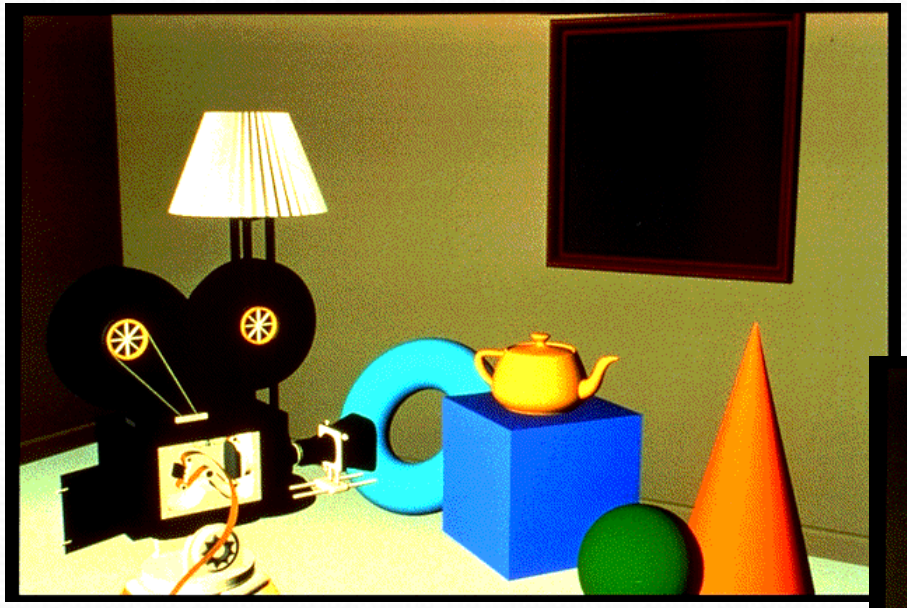
- Durante a rasterização dos polígonos, que correspondem às projeções das faces sobre a tela virtual, as intensidades são interpoladas ao longo de cada aresta e ao longo dos segmentos de varredura

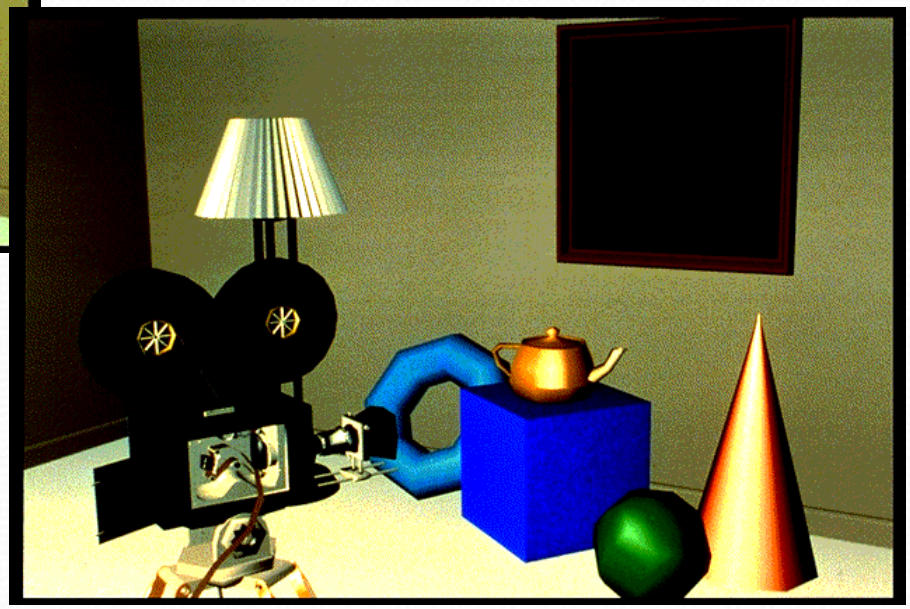


Colorização de Gouraud



Colorização de Gouraud

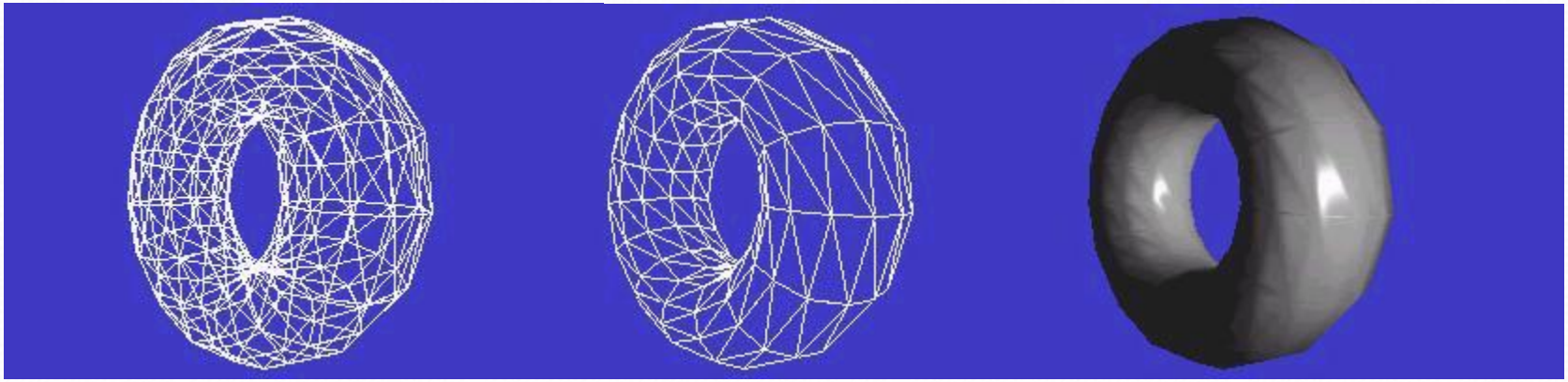




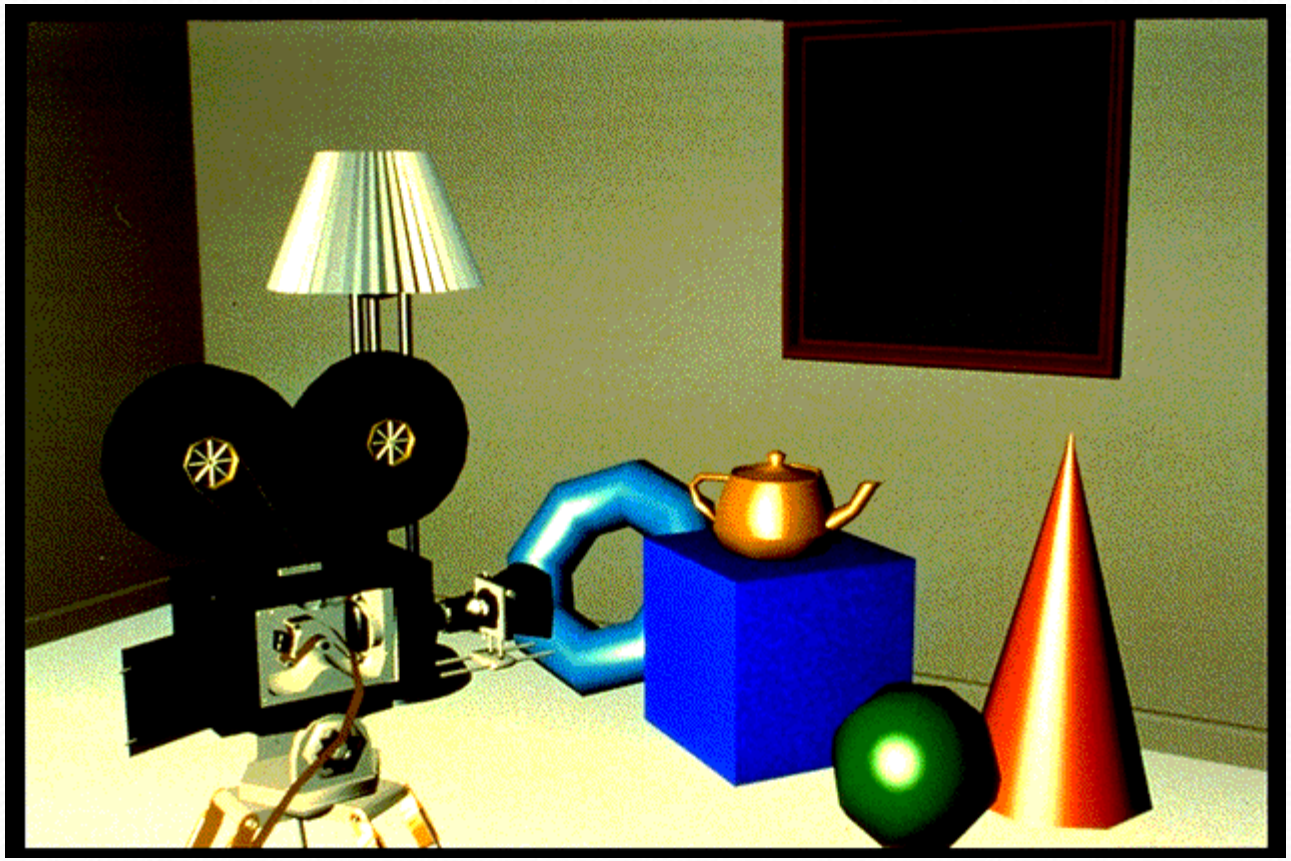
Colorização de Phong

- É também chamada de Phong shading.
- É um modelo de iluminação local simples.
- Este método, ao invés de interpolar as intensidades, interpola diretamente as normais durante a rasterização.
- Os resultados obtidos são excelentes, porém paga-se com um alto custo computacional.

Colorização de Phong



Colorização de Phong



Visibilidade

Conceitos Teóricos e Práticos

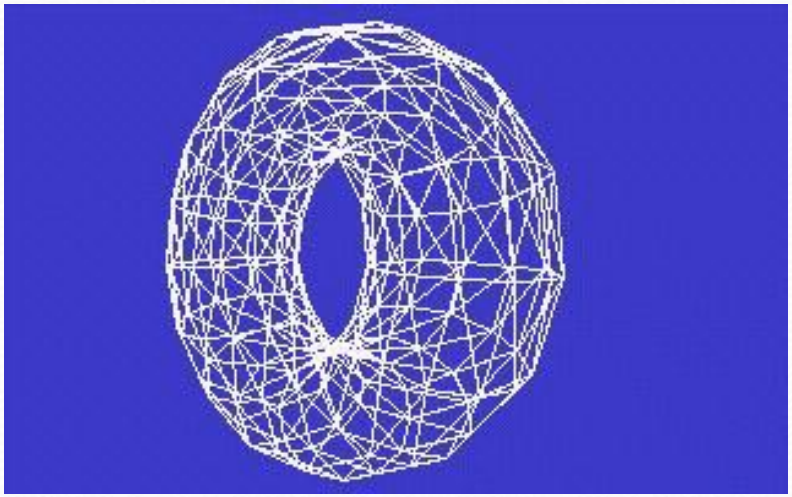
O Problema de Visibilidade

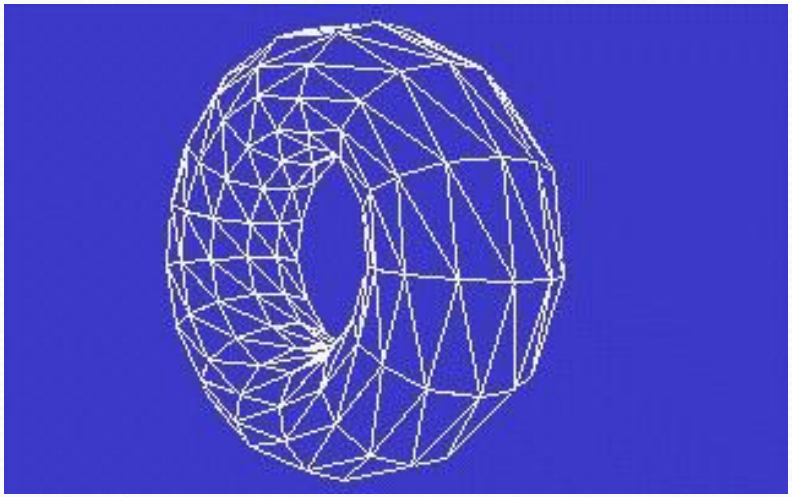
- Numa cena tri-dimensional, normalmente não é possível ver todas as superfícies de todos os objetos
- Não queremos que objetos ou partes de objetos não visíveis apareçam na imagem
- Problema importante que tem diversas ramificações
  - Descartar objetos que não podem ser vistos (*culling*)
  - Recortar objetos de forma a manter apenas as partes que podem ser vistas (*clipping*)
  - Desenhar apenas partes visíveis dos objetos
    - Em aramado (*hidden-line algorithms*)
    - Superfícies (*hidden surface algorithms*)
  - Sombras (visibilidade a partir de fontes luminosas)

Visibilidade

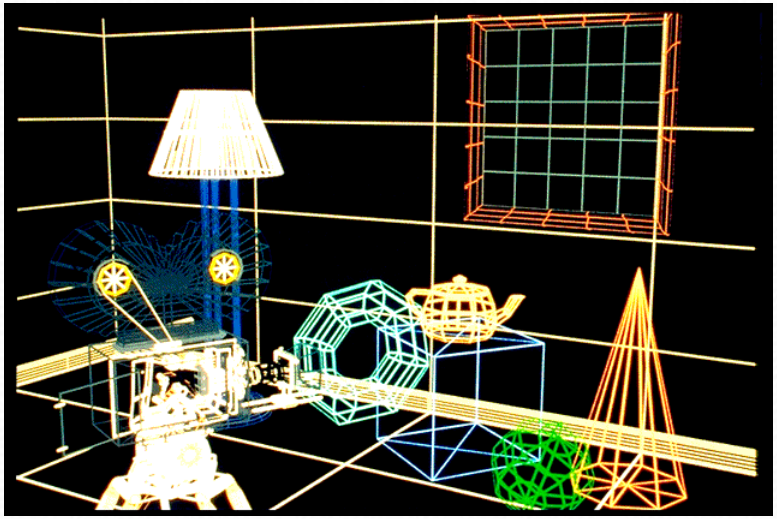
- Algoritmos para eliminação de superfícies ocultas ou escondidas.
- O problema é determinar quais as sub-partes da cena que estão dentro do espaço de visão da câmera e que são visíveis ao serem projetadas.

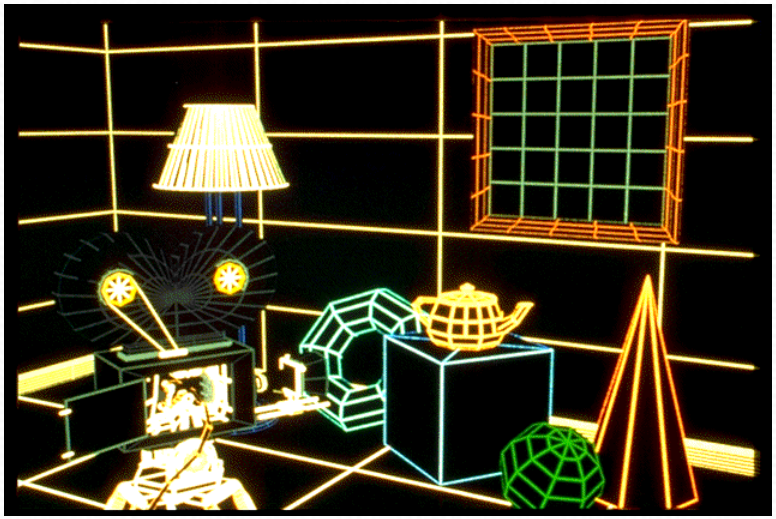
Visibilidade





Visibilidade





Visibilidade

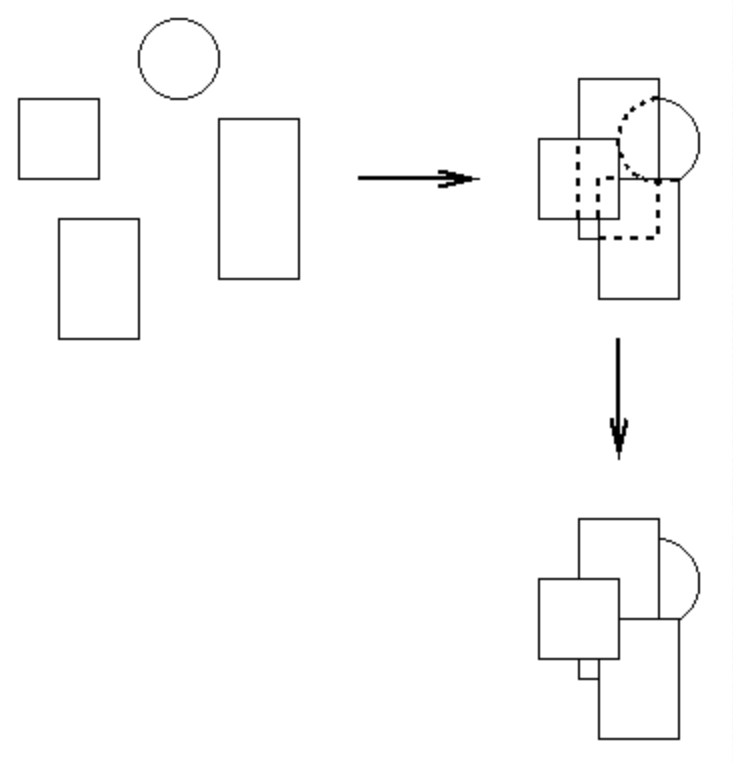
- Os algoritmos de visibilidade podem ser interpretados como: Rendering = Visibilidade + Shading
  - Visibilidade: o que é visível ?
  - Shading: qual a sua côr ?
- A transformação de visualização deve ser eficiente o suficiente, de modo a determinar a visibilidade levando em conta a perspectiva.

Visibilidade

- A determinação da visibilidade pode estar associada à algum algoritmo de rasterização
- Uma vez determinada a visibilidade, deve-se calcular a função de iluminação que determinará a côr de cada elemento da imagem (pixel).
- Em função da natureza da imagem sendo gerada, existem algoritmos de visibilidade: 2D E 3D.

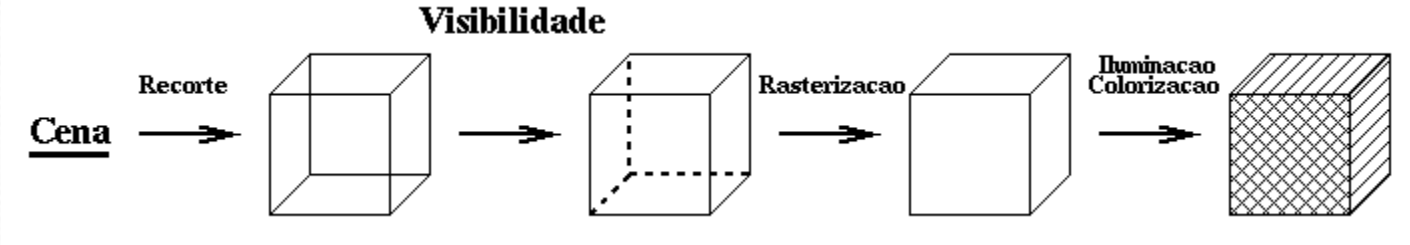
Visibilidade

- 2D, também conhecidos como algoritmos de linhas escondidas



Visibilidade

- 3D, também conhecidos como algoritmos de superfícies escondidas



Espaço do Objeto x Espaço da Imagem

- Métodos que trabalham no espaço do objeto
  - Entrada e saída são dados geométricos
  - Independente da resolução da imagem
  - Menos vulnerabilidade a *aliasing*
  - Rasterização ocorre *depois*
  - Exemplos:
    - Maioria dos algoritmos de recorte e *culling*
      - Recorte de segmentos de retas
      - Recorte de polígonos
    - Algoritmos de visibilidade que utilizam recorte
      - Algoritmo do pintor
      - BSP-trees
      - Algoritmo de recorte sucessivo
      - Volumes de sombra

Classificação dos Algoritmos

- O problema fundamental é a determinação das superfícies mais próximas do observador.
- Este problema envolve uma ordenação parcial, ou seja, até a primeira superfície opaca em cada vizinhança da imagem
  - Exemplos de algoritmos de ordenação
    - Quicksort
    - Bubblesort
    - Heapsort
    - Mergesort

Espaço do Objeto x Espaço da Imagem

- Métodos que trabalham no espaço da imagem
  - Entrada é vetorial e saída é matricial
  - Dependente da resolução da imagem
  - Visibilidade determinada apenas em pontos (pixels)
  - Podem aproveitar aceleração por hardware
  - Exemplos:
    - Z-buffer
    - Algoritmo de Warnock
    - Mapas de sombra

Visibilidade

Algoritmos

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?

Complexidade do Problema

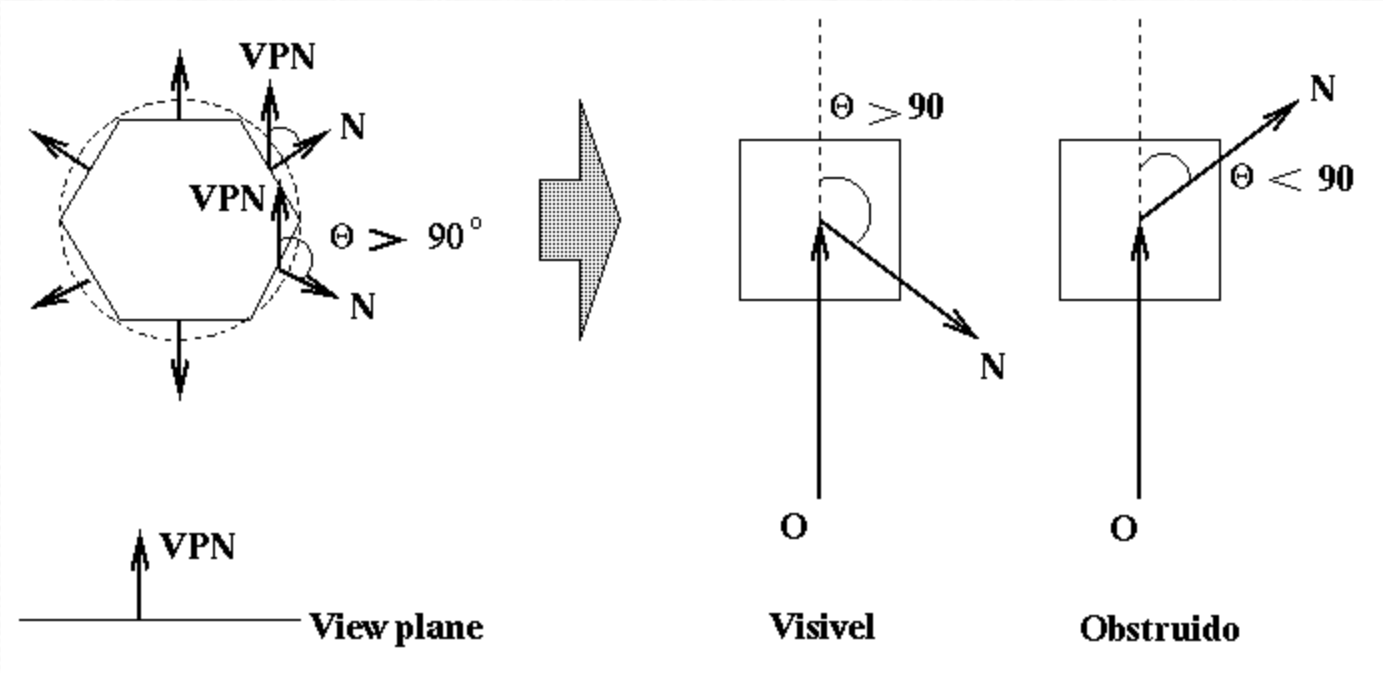
- Fatores que influenciam o problema
  - Número de pixels
    - Em geral procura-se minimizar o número total de pixels pintados
    - Resolução da imagem / *depth* buffer
    - Menos importante se rasterização é feita por hardware
  - Número de objetos
    - Técnicas de "*culling*"
    - Células e portais
    - Recorte pode aumentar o número de objetos

BackFace Culling

- Se a superfície de um objeto é aproximada por uma superfície poligonal, então as faces são polígonos e englobam completamente o volume do objeto.
- Assume-se que:
  - As normais de todas as faces do objeto (sólido) apontam para fora do volume
  - As faces, cujas normais apontam na direção contrária ao observador, estão numa parte do poliedro cuja visibilidade é bloqueada por outras faces mais próximas ao observador (fazem um ângulo menor que 900).

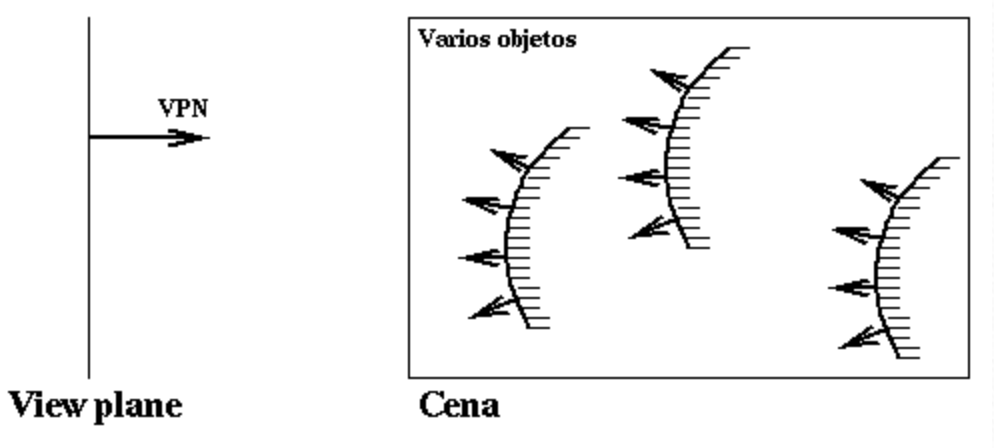
BackFace Culling

- As faces cujas normais fazem um ângulo menor que 900 estão numa parte do poliedro cuja visibilidade é bloqueada por outras faces.



BackFace Culling

- E se a cena possuir mais de 1 objeto ?
- E se o objeto fôr côncavo ?

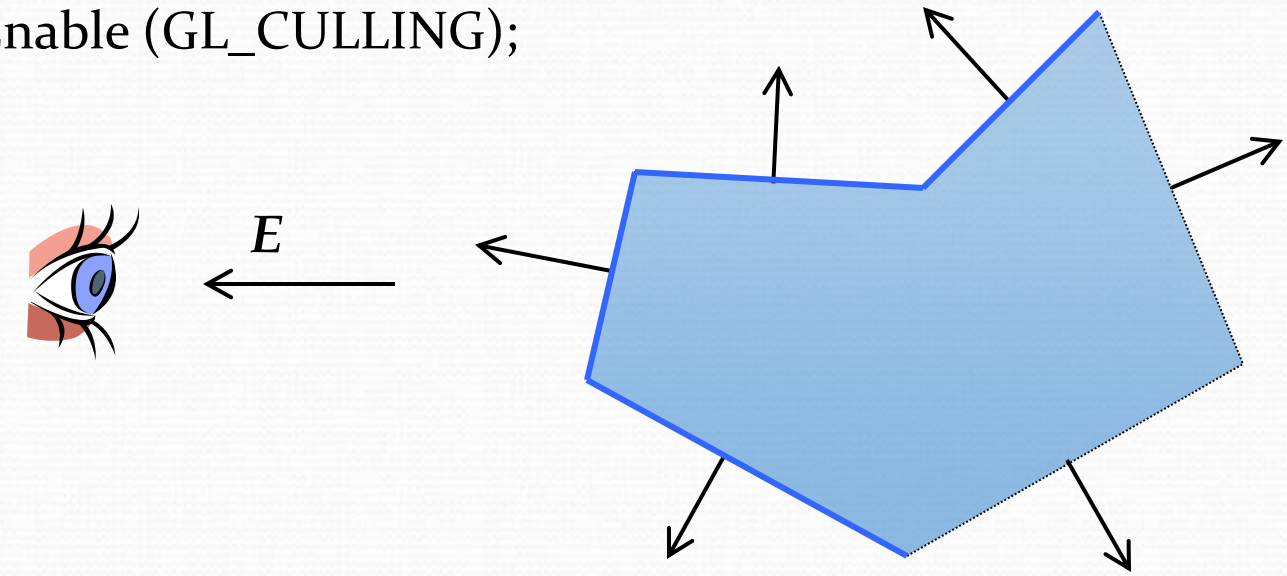


BackFace Culling

- Um projetor que atravessa um sólido intercepta o mesmo número de faces visíveis e obstruídas Portanto, o método de backface culling reduz à metade o número de polígonos que precisam ser considerados pelos métodos de visibilidade com precisão de imagem Vantagem
  - Velocidade: aumenta por um fator de "2"
- Desvantagem
  - Não é suficiente por si só

Backface Culling em OpenGL

- 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
- OpenGL
  - glEnable (GL_CULLING);



Z-Buffer

- É o algoritmo de visibilidade mais simples de ser implementado.
- Pode ser feito em software ou em hardware (placas aceleradoras gráficas) O algoritmo exige 2 buffers para a imagem:
    - Frame buffer, que armazena os valores de côr para cada pixel.
    - Z buffer, com o mesmo número de entradas, que armazena os valores de profundidade para cada pixel (é às vezes chamado de range image ou depth image ou depth buffer)

Z-Buffer

- Método que opera no espaço da imagem
- 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 → GLUT
    - 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

    glDepthFunc(...)

Z-Buffer

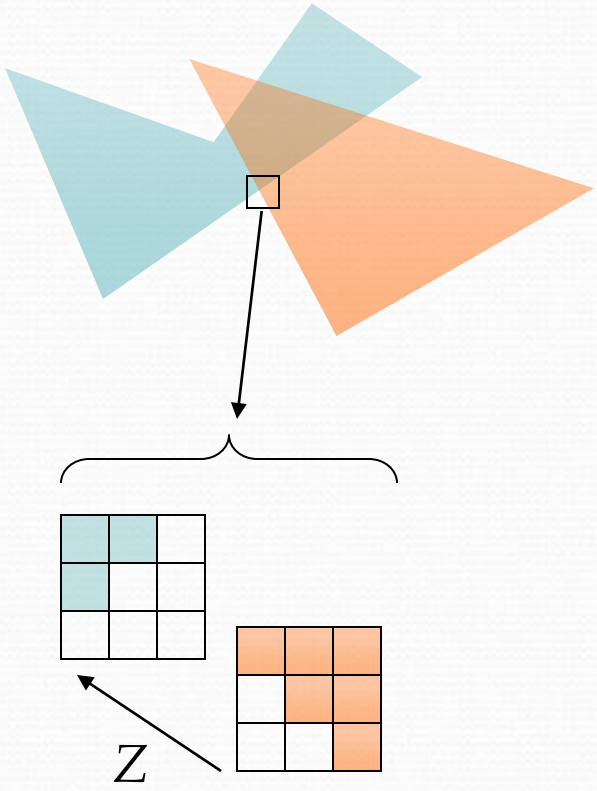
- Vantagens:
  - Simples e comumente implementado em Hardware
  - Objetos podem ser desenhados em qualquer ordem
- Desvantagens:
  - Rasterização independe de visibilidade
    - Lento se o número de polígonos é grande
  - Erros na quantização de valores de profundidade podem resultar em imagens inaceitáveis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

Z-Buffer e Transparência

- Se há objetos semi-transparentes, a ordem de renderização é importante
- Após a renderização de um objeto transparente, atualiza-se o z-buffer?
  - Sim → novo objeto por trás não pode mais ser renderizado
  - Não → z-buffer fica incorreto
- Soluções
  - Estender o z-buffer → A-buffer
  - Pintar de trás para frente → Algoritmo do pintor
    - Necessário de qualquer maneira, para realizar transparência com *blending* (canal alfa)

A-Buffer

- Melhoramento da idéia do z-buffer
- Permite implementação de transparência e de filtragem (*anti-aliasing*)
- Para cada pixel manter lista ordenada por z onde cada nó contém
  - Máscara de subpixels ocupados
  - Cor ou ponteiro para o polígono
  - Valor de z (profundidade)



A-Buffer

- Fase 1: Polígonos são rasterizados
  - Se pixel completamente coberto por polígono e polígono é opaco
    - Inserir na lista removendo polígonos mais profundos
  - Se o polígono é transparente ou não cobre totalmente o pixel
    - Inserir na lista
- Fase 2: Geração da imagem
  - Máscaras de subpixels são misturadas para obter cor final do pixel

A-Buffer

- Vantagens
  - Faz mais do que o z-buffer
  - Idéia da máscara de subpixels pode ser usada com outros algoritmos de visibilidade
- Desvantagens
  - Implementação (lenta) por software
  - Problemas do z-buffer permanecem
    - Erros de quantização em z
    - Todos os polígonos são rasterizados

Exercício Prático

1. Vamos implementar em OpenGL o algoritmo de ZBuffer. Para isso desenhe um cubo e coloque para habilitar/desabilitar o algoritmo. Veja o que acontece.
2. Vamos implementar em OpenGL o algoritmo de BackFaceCulling. Para isso desenhe um cubo e coloque para habilitar/desabilitar o algoritmo. Veja o que acontece.