Inatel

C209 – Computação Gráfica e Multimídia

Realismo Visual e Iluminação
Parte 1: Visibilidade

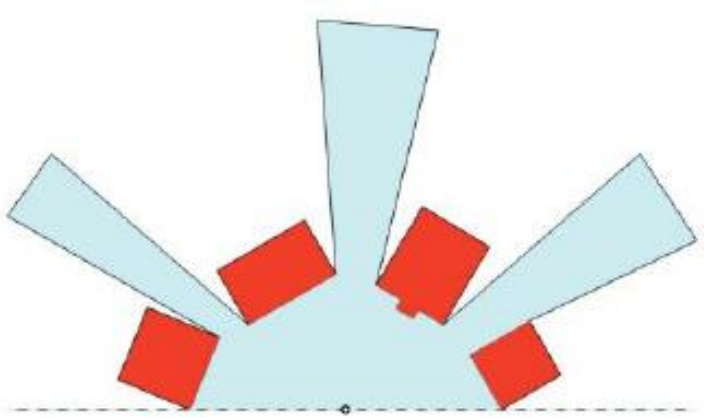
Marcelo Vinícius Cysneiros Aragão

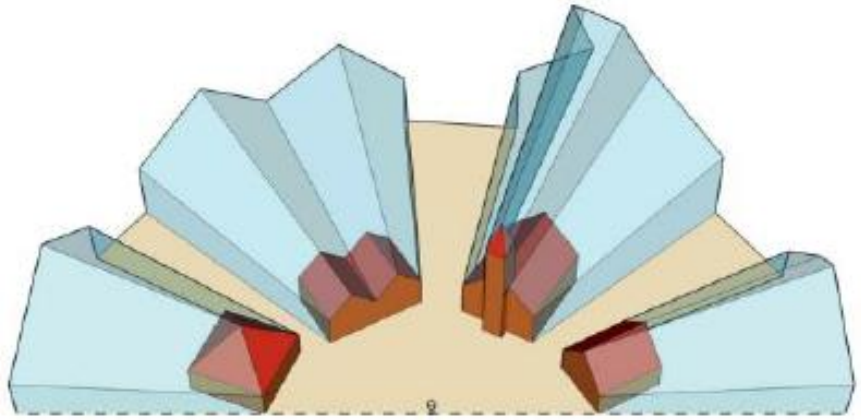
marcelovca90@inatel.br

Inatel
Instituto Nacional de Telecomunicações

O Problema de Visibilidade

- Numa cena tridimensional, 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.





O Problema de Visibilidade

- 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)

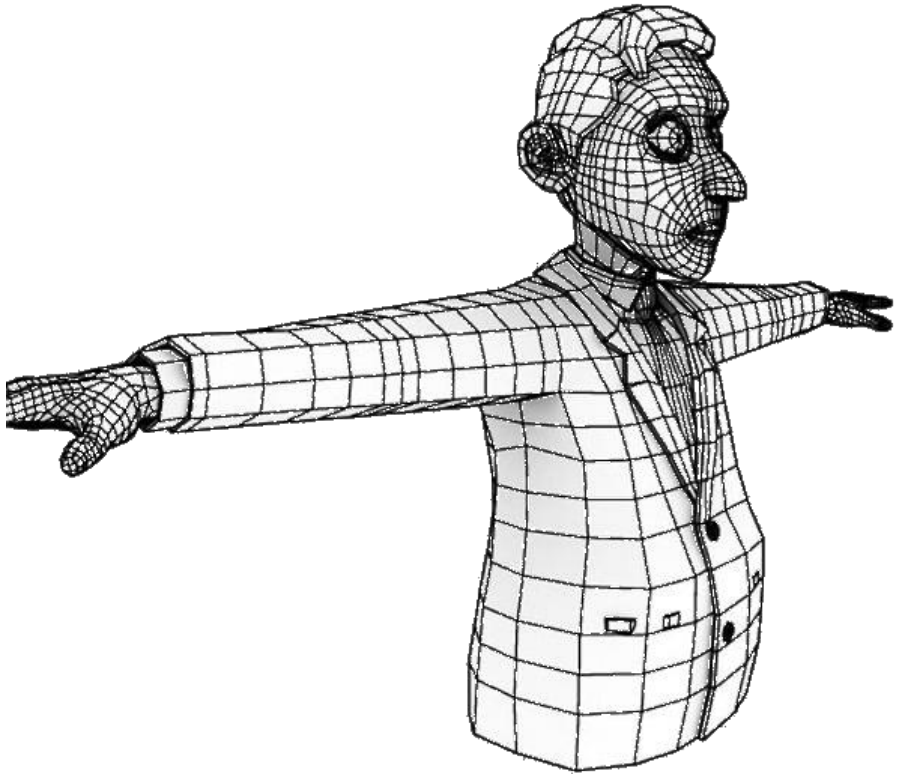
Motivação

- Dispositivos matriciais sobrescrevem os objetos (aparecem os desenhados por último, quando há sobreposição).
  - Em 3D, se nada for feito para corrigir a ordem de desenho, a imagem é gerada incorretamente,
- Os algoritmos de visibilidade estruturam os objetos da cena, de modo a que sejam exibidos corretamente.

Inatel
Instituto Nacional de Telecomunicações

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 software ou hardware?



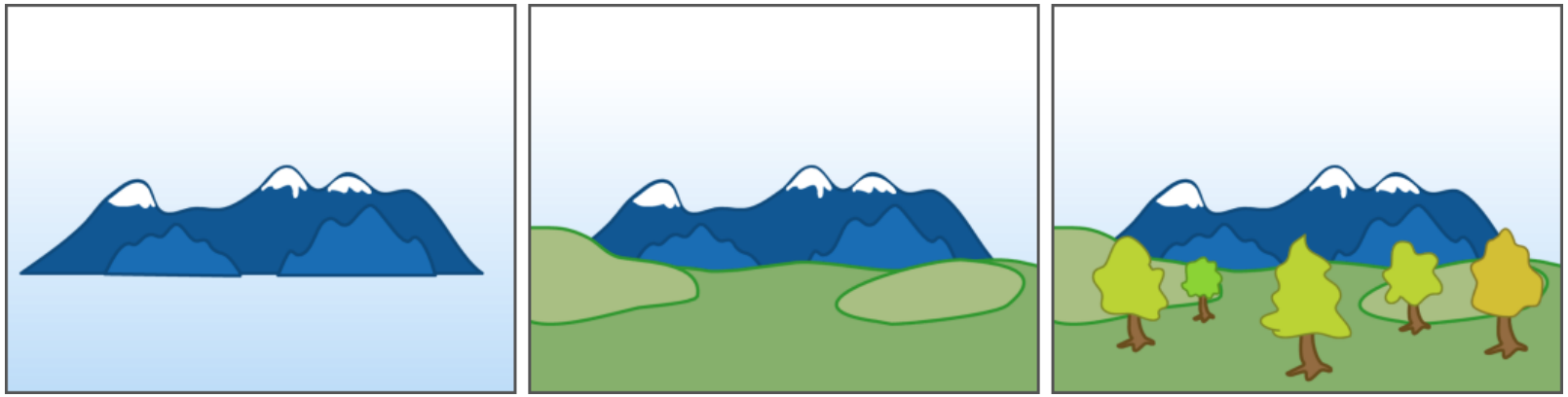
Complexidade do Problema

- 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

Inatel
Instituto Nacional de Telecomunicações

Algoritmo do Pintor

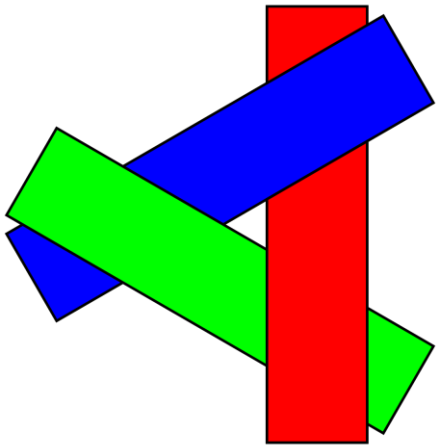
- Também conhecido como prioridade em *Z* (*depth priority*)
- Ideia é pintar objetos mais distantes (*background*) antes de pintar objetos próximos (*foreground*)



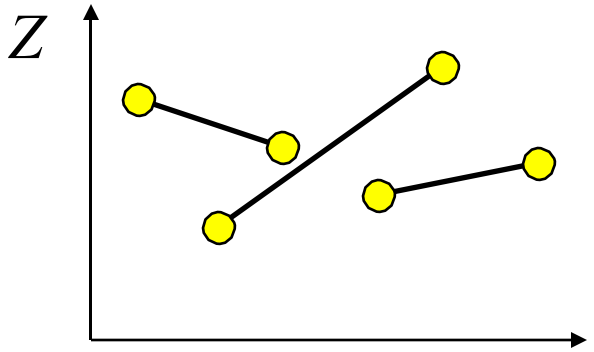
As montanhas distantes são pintadas primeiro, seguidas pelos campos próximos; finalmente, os objetos mais próximos nessa cena - as árvores - são pintadas.

Algoritmo do Pintor

- Requer que objetos sejam ordenados em *Z*
  - Complexidade O (*N* log *N*)
  - Pode ser complicado em alguns casos (vide imagem à esquerda)
  - Na verdade, a ordem não precisa ser total se projeções dos objetos não se interceptam (vide imagem à direita)
  - **Vantagem:** não faz uso de máscara de pixels já pintados
  - **Desvantagem:** um mesmo pixel é pintado diversas vezes



Não há ordem possível

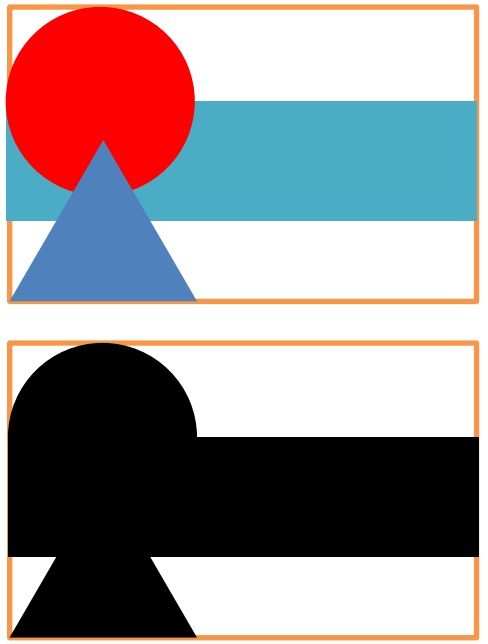


Que ponto usar para determinar ordem?

Inatel
Instituto Nacional de Telecomunicações

Algoritmo Reverso do Pintor

- 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*
- **Vantagem:** cada pixel é pintado somente uma vez
- **Desvantagem:** máscara pode tornar-se complexa

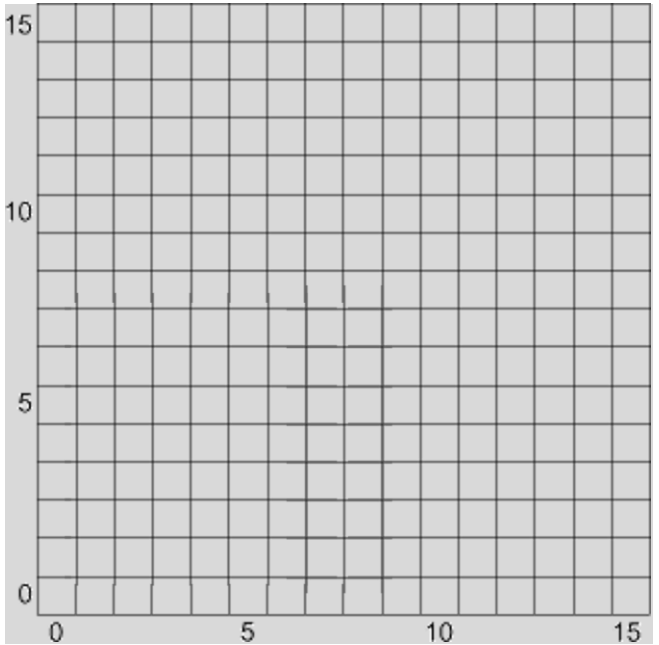


Z-Buffer

- 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

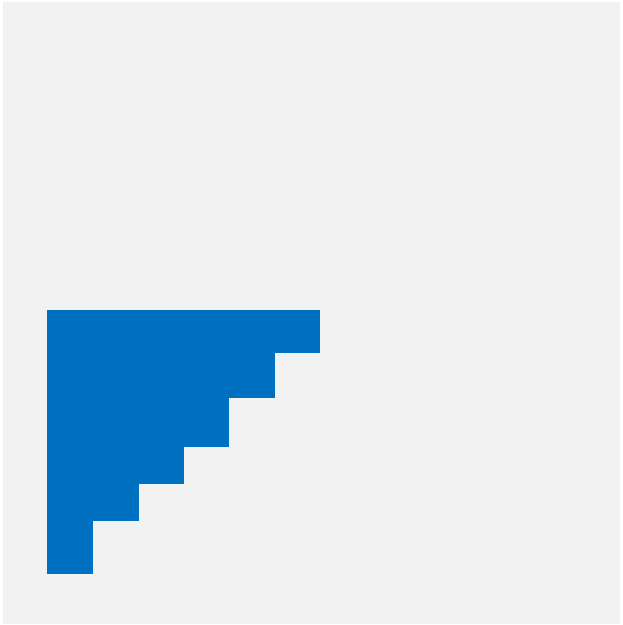
Z-Buffer

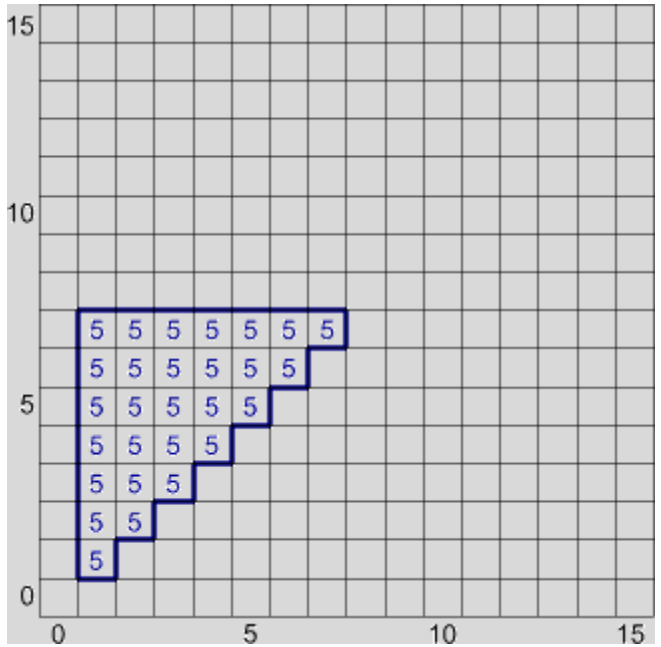
- Exemplo:
  - *Desenhar os seguintes objetos:*
    - 1º Polígono: (1,1,5) , (7,7,5), (1,7,5)
    - 2º Polígono: (4, 5, 9), (10, 5, 9), (10, 9, 9), (4, 9, 9)
    - 3º Polígono: (2, 8, 3), (2, 3, 8), (7, 3, 3)



Z-Buffer

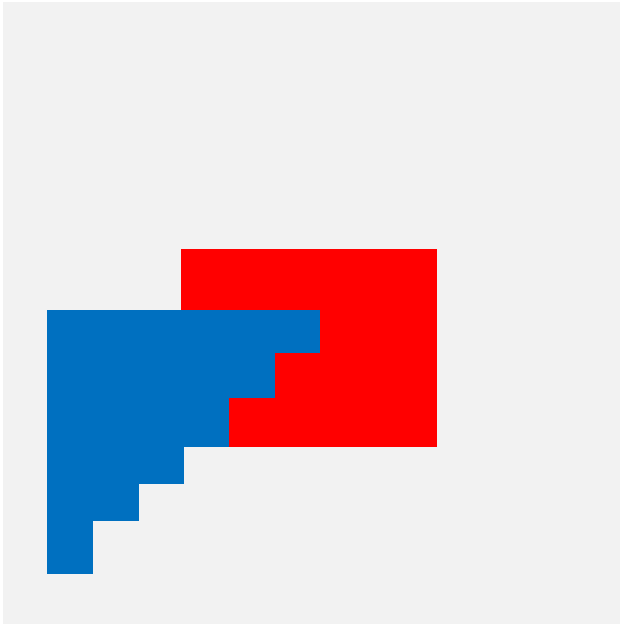
- Exemplo:
  - *Desenhar os seguintes objetos:*
    - 1º Polígono: (1,1,5) , (7,7,5), (1,7,5)
    - 2º Polígono: (4, 5, 9), (10, 5, 9), (10, 9, 9), (4, 9, 9)
    - 3º Polígono: (2, 8, 3), (2, 3, 8), (7, 3, 3)

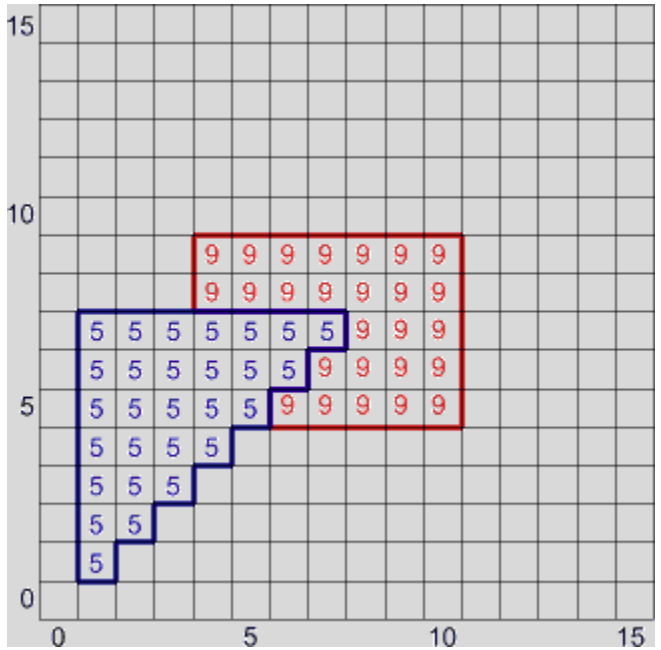




Z-Buffer

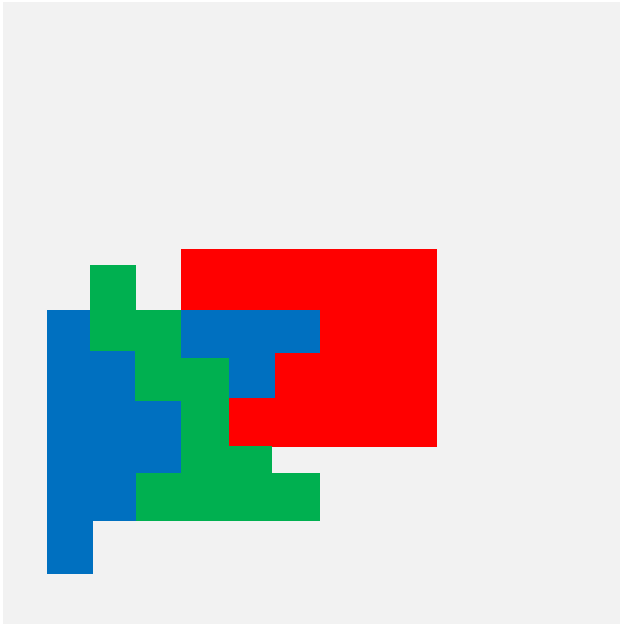
- Exemplo:
  - *Desenhar os seguintes objetos:*
    - 1º Polígono: (1,1,5) , (7,7,5), (1,7,5)
    - 2º Polígono: (4, 5, 9), (10, 5, 9), (10, 9, 9), (4, 9, 9)
    - 3º Polígono: (2, 8, 3), (2, 3, 8), (7, 3, 3)

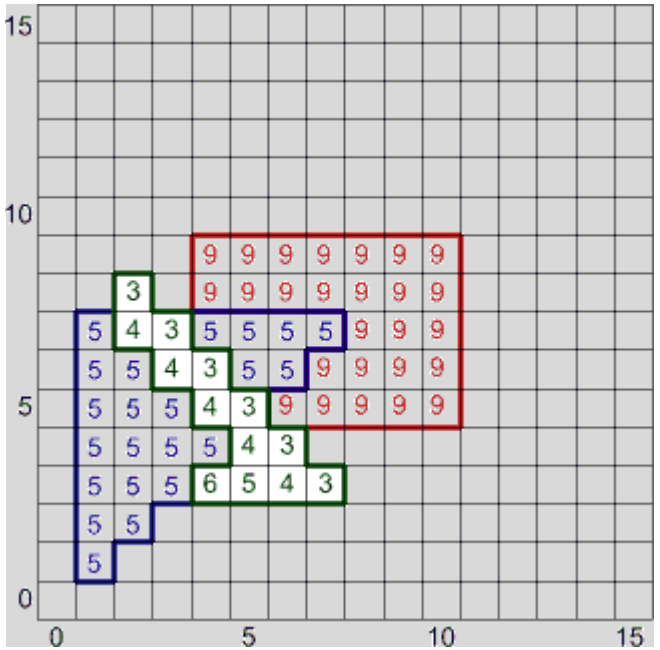




Z-Buffer

- Exemplo:
  - *Desenhar os seguintes objetos:*
    - 1º Polígono: (1,1,5) , (7,7,5), (1,7,5)
    - 2º Polígono: (4, 5, 9), (10, 5, 9), (10, 9, 9), (4, 9, 9)
    - 3º Polígono: (2, 8, 3), (2, 3, 8), (7, 3, 3)





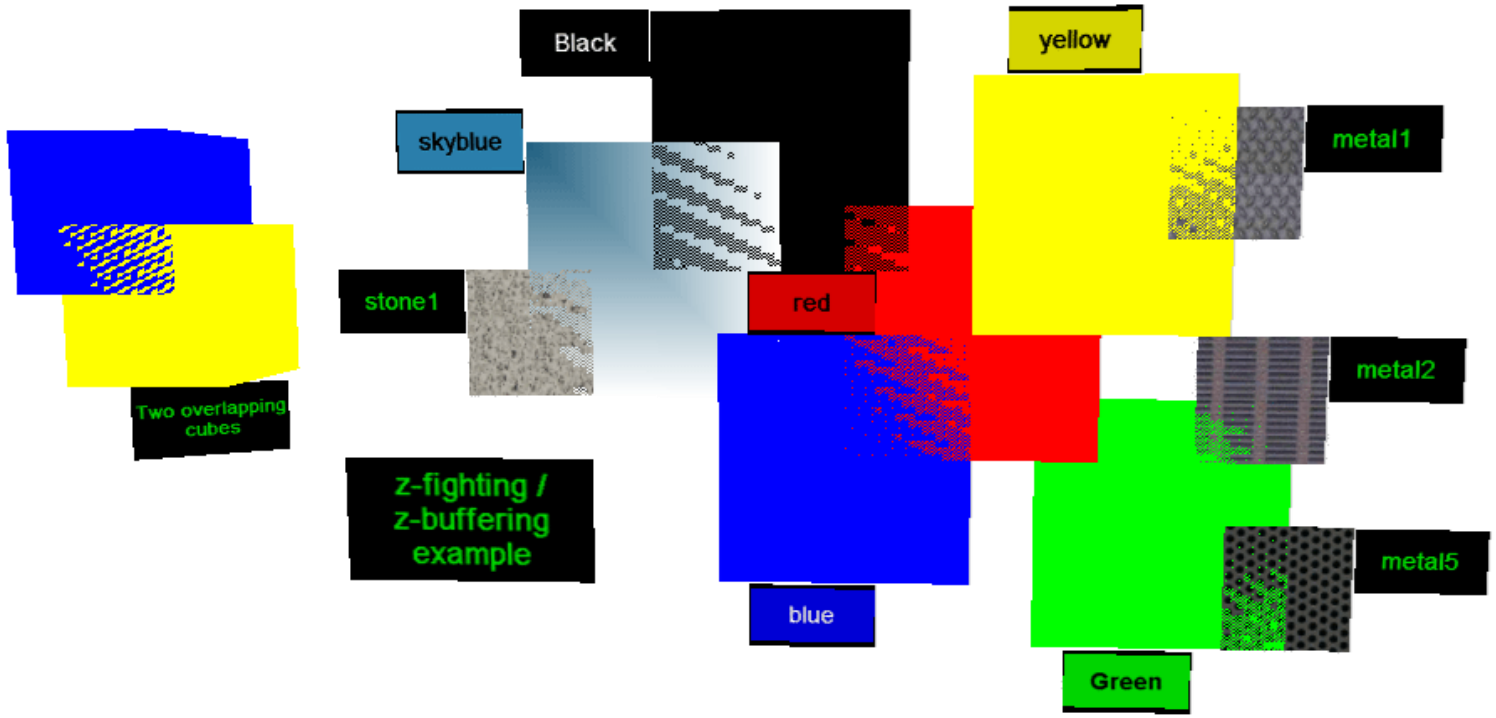
Z-Buffer

- 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 → Stitching
  - Dificulta o uso de transparência ou técnicas de anti- serrilhamento
    - É preciso ter informações sobre os vários polígonos que cobrem cada pixel

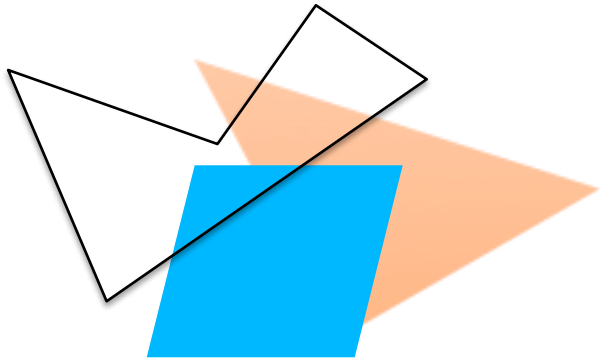
Z-Buffer e Z-Fighting (Stitching)

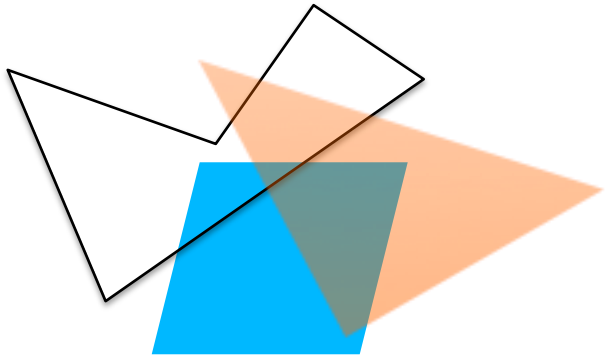
- É um fenômeno que ocorre quando 2 ou mais primitivas têm valores próximos (ou iguais) no z-buffer.
- Ocorre muito entre polígonos coplanares, onde duas faces ocupam essencialmente o mesmo espaço.
- Os pixels afetados são renderizados com fragmentos de um polígono ou o outro arbitrariamente, de acordo com a precisão do z-buffer.



Z-Buffer e Transparência

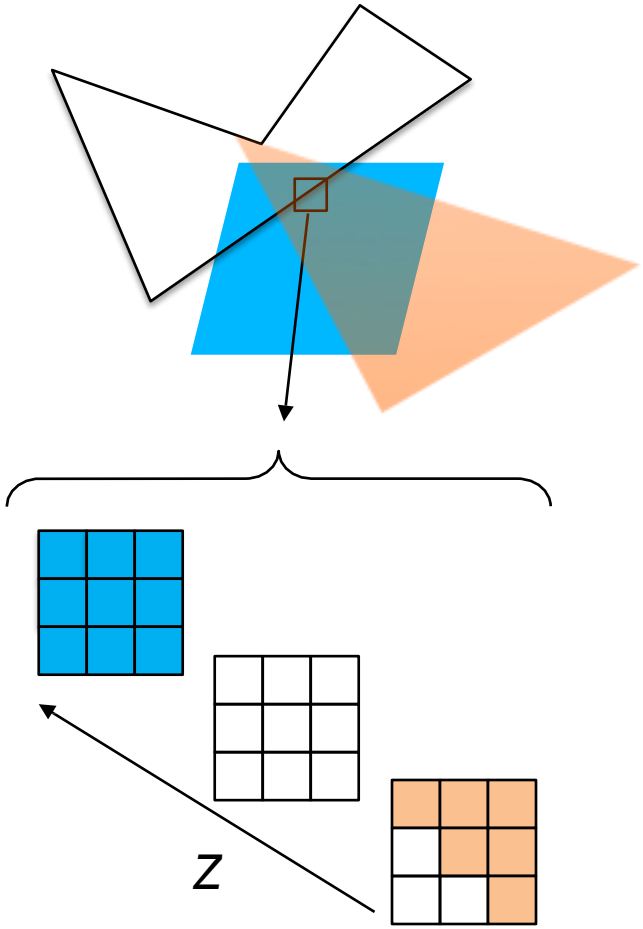
- Se há objetos semi-transparentes, a ordem de renderização é importante
- Após a renderização de um objeto transparente, atualiza-se o z-buffer?
  - Sim → novo objeto por trás não pode mais ser renderizado
  - Não → z-buffer fica incorreto
- Soluções
  - Estender o z-buffer → A-buffer
  - Pintar de 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
  - Ideia 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

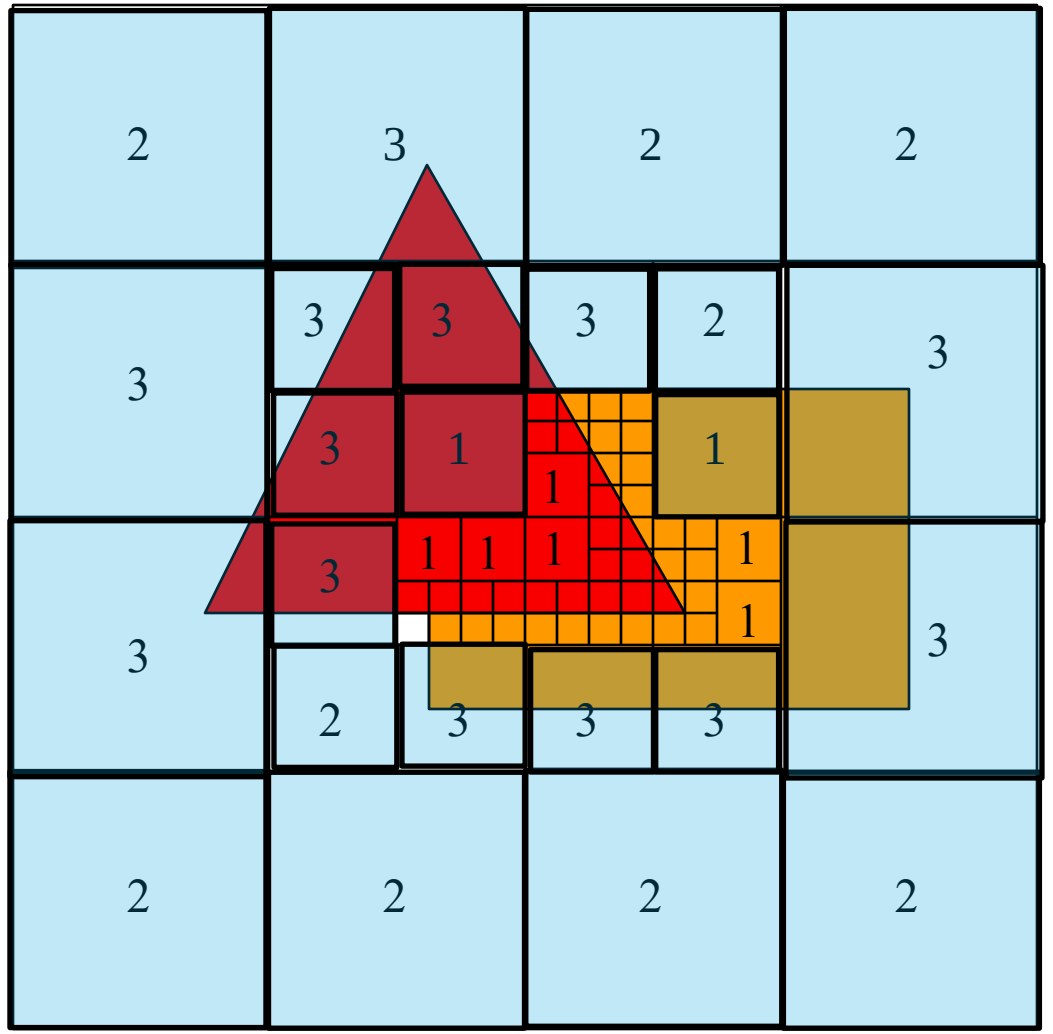
Inatel
Instituto Nacional de Telecomunicações

Algoritmo de Warnock

- Subdivide o espaço da imagem para explorar coerência de área
- Sabemos como pintar uma determinada sub-região da imagem se:
  1. Um polígono cobre a região totalmente e em toda região é mais próximo que os demais
  2. Nenhum polígono a intercepta
  3. Apenas um polígono a intercepta
- Se a sub-região não satisfaz nenhum desses critérios, é subdividida recursivamente à maneira de uma quadtree
  - Se sub-região se reduz a um pixel, pintar o polígono com menor profundidade

Inatel
Instituto Nacional de Telecomunicações

Algoritmo de Warnock



Algoritmo de Warnock

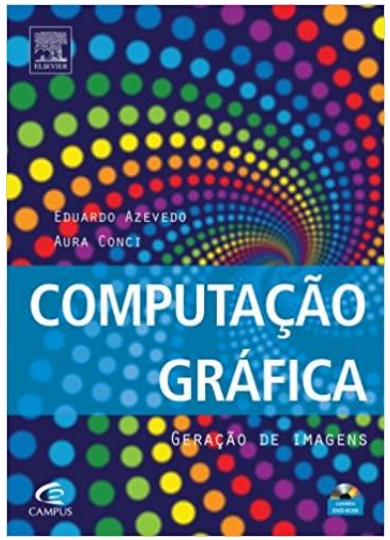
- Vantagens
  - Explora coerência de área
    - Apenas áreas que contêm arestas precisam ser subdivididas até o nível de pixel
  - Pode ser adaptado para suportar transparência
  - Levando a recursão até tamanho de subpixel, pode-se fazer filtragem de forma elegante
  - Pinta cada pixel apenas uma vez
- Desvantagens
  - Testes são lentos
  - Aceleração por hardware improvável

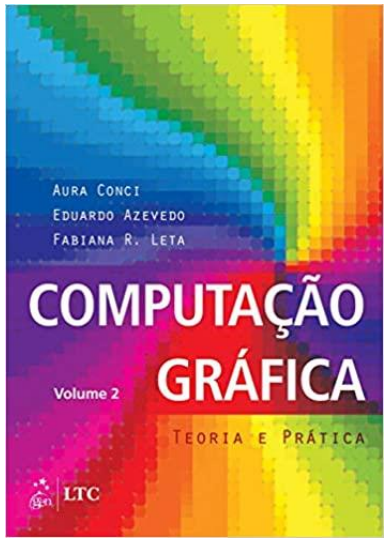
Inatel
Instituto Nacional de Telecomunicações

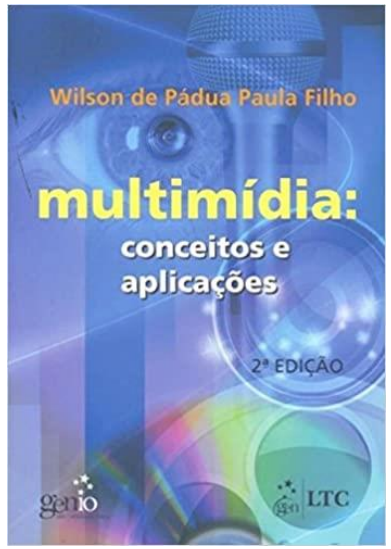
Pseudocódigos

- Z-Buffer
  - https://raw.githubusercontent.com/marcelovca90-inatel/C209/master/pseudocodigos/z-buffer.txt
- A-Buffer
  - https://raw.githubusercontent.com/marcelovca90-inatel/C209/master/pseudocodigos/a-buffer.txt
- Warnock
  - https://raw.githubusercontent.com/marcelovca90-inatel/C209/master/pseudocodigos/warnock.txt

Referências & Links Interessantes







AZEVEDO, Eduardo; CONCI, Aura, Computação gráfica volume 1: geração de imagens. Rio de Janeiro, RJ. Editora Campus, 2003, 353 p. ISBN 85-352-1252-3.

AZEVEDO, Eduardo; CONCI, Aura; LETA, Fabiana R. Computação gráfica volume 2: teoria e prática. Rio de Janeiro, RJ: Editora Elsevier, 2007, 384 p. ISBN 85-352-2329-0.

PAULA FILHO, Wilson de Pádua, Multimídia: Conceitos e aplicações. Rio de Janeiro, RJ: LTC, 2000, 321 p. ISBN 978-85-216-1222-3.