Preenchimento de Polígonos

Uéliton Freitas

Universidade Católica Dom Bosco - UCDB

freitas.ueliton@gmail.com

6 de novembro de 2014

Sumário

Introdução

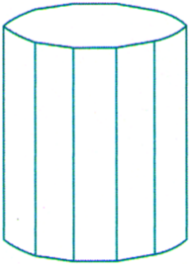
Preenchimento de Polígonos

- Além do desenho de linhas, uma outra construção útil é o **preenchimento de áreas**.
  - Usado para descrever superfícies ou objetos sólidos.
- Embora qualquer forma possa ser preenchida, normalmente as **APIs gráficas suportam polígonos**.
  - Maior eficiência por serem descritos por equações lineares.
  - Maioria das superfícies curvas podem ser aproximadas por polígonos.

Introdução

Preenchimento de Polígonos

- Aproximação de curva é normalmente chamada de **tesselação** de uma superfície ou **malha de polígonos**.
- Estas aproximações podem ser rapidamente geradas como visões wire-frame.



Introdução

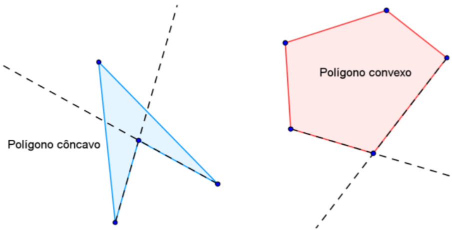
Preenchimento de Polígonos

- Um polígono é uma **figura plana** especificada por um conjunto de 3 ou mais **vértices**, **ligados** sequencialmente por **arestas**(linhas).
- Arestas possuem pontos em comum somente em seu ponto inicial e final.
- Todos os vértices estão no mesmo plano.
- Devido a erros de arredondamento, as arestas de um polígono podem são ser coplanares.
  - Utiliza-se triângulos para resolver este problema.

Introdução

Classificação de Polígonos

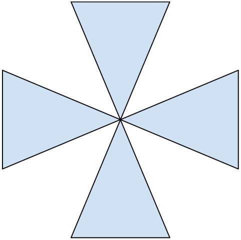
- Se todos os ângulos interiores de um polígono forem menores que 180°, o polígono é **convexo** caso contrário é **côncavo**.
- Em um polígono convexo, dois pontos interiores definem um segmento de reta também no interior.



Introdução

Classificação de Polígonos

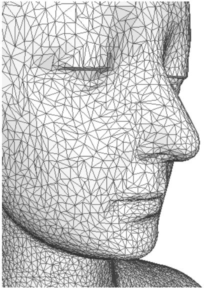
- O termo polígono **degenerado** descreve um polígono com vértices colineares, ou que apresentam vértices repetidos.
- Uma **API** gráfica para ser **robusta** deve **regeitar** polígonos **não planares ou degenerados**.
  - Na verdade isso é deixado a cargo do programador.



Introdução

Classificação de Polígonos

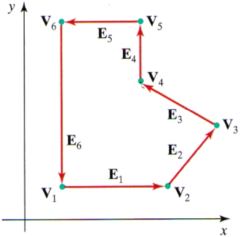
- **APIs** gráficas trabalham apenas com com **polígonos convexos**.
  - Melhor dividir um polígono côncavo em um conjunto de polígonos convexos.
  - **OpenGL** requer que todos os polígonos sejam convexos.



Introdução

Identificando Polígonos Côncavos

- Cria-se vetores com as arestas e faz-se o produto vetorial sobre arestas adjacentes.
  - A **coordenada-z** de todos os produtos **devem ter o mesmo sinal** em um polígono **convexo**.

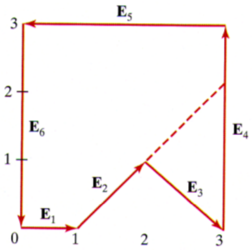


$$(\mathbf{E}_1 \times \mathbf{E}_2)_z > 0$$
$$(\mathbf{E}_2 \times \mathbf{E}_3)_z > 0$$
$$(\mathbf{E}_3 \times \mathbf{E}_4)_z < 0$$
$$(\mathbf{E}_4 \times \mathbf{E}_5)_z > 0$$
$$(\mathbf{E}_5 \times \mathbf{E}_6)_z > 0$$
$$(\mathbf{E}_6 \times \mathbf{E}_1)_z > 0$$

Introdução

Dividindo Polígonos Côncavos

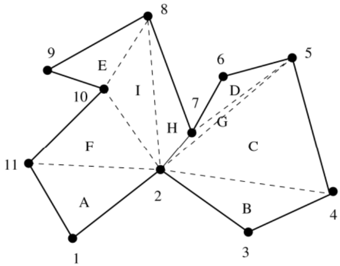
- Cria-se vetores para dois vértices consecutivos.
  - $E_k = V_{k+1} - V_k$
- Calcular o produto vetorial destes no sentido anti-horário.
- Se algum produto for negativo, o polígono é côncavo
  - Dividindo o polígono ao longo da linha do primeiro vértice.



Introdução

Dividindo Polígonos em Triângulos

- Um **polígono convexo** pode ser divididos em **triângulos**.
    - Pegue quaisquer três vértices consecutivos no sentido anti-horário e forme um triângulo.
    - Caso as arestas do triângulo não cruze nenhuma outra aresta do polígono, retire o vértice da lista de vértices.
    - Repita o procedimento até que sobrem apenas três vértices;



Teste de Interior e Exterior

Teste de Interior e Exterior

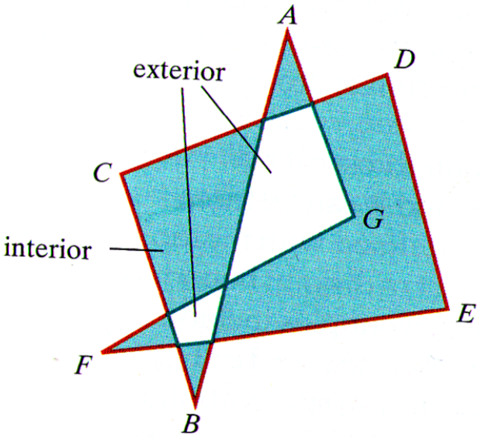
- Vários processos gráficos precisam **identificar regiões interiores** de objetos mais **complexos** que quadrados e círculos.
- Serão apresentados dois algoritmos:
  - Regra **par-ímpar**(ou regra da paridade ímpar).
  - Regra do **winding-number** não zero.

Teste de Interior e Exterior

Regra Par-Impar

- Desenhar um segmento de reta de um ponto P a um ponto distante e fora dos limites das coordenadas do polígono.
- Contar os cruzamentos de arestas com esse segmento.
  - Se o número for **ímpar**, P está dentro.
  - Caso contrário P está **fora**.
- Deve-se assegurar que o segmento de reta não intercepte nenhum vértice do polígono.

Teste de Interior e Exterior



Teste de Interior e Exterior

Regra do Winding-Number não-zero

- Conta o número de **vezes** que a **fronteira** de um objeto **gira** em volta de um ponto particular na **direção anti-horária**
    - Um ponto é dito **interior** se sua contagem for **diferente de zero**.

Teste de Interior e Exterior

Algoritmo - Regra do Winding-Number não-zero

- Inicia-se a contagem com zero.

Teste de Interior e Exterior

Algoritmo - Regra do Winding-Number não-zero

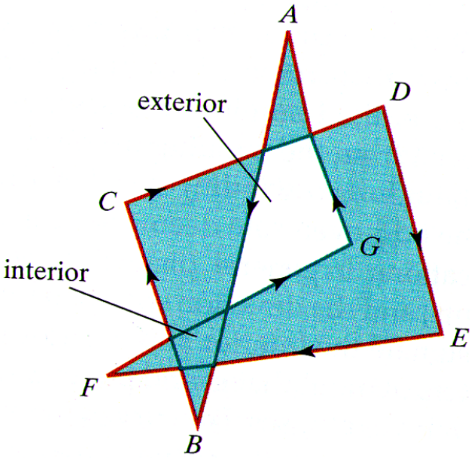
- Inicia-se a contagem com zero.
- Defini-se um segmento de reta de uma posição P até um ponto distante.
  - Não pode passar pelo vértice.

Teste de Interior e Exterior

Algoritmo - Regra do Winding-Number não-zero

- Inicia-se a contagem com zero.
- Defini-se um segmento de reta de uma posição P até um ponto distante.
  - Não pode passar pelo vértice.
- Conta a quantidade de cruzamentos com as arestas (direcionais)
  - $+1$ toda vez que cruzar uma aresta da **direita para esquerda**.
  - -1 toda vez que cruzar uma aresta da **esquerda para direita**.

Teste de Interior e Exterior



Teste de Interior e Exterior

Regra do Winding-Number não-zero
Processo Baseado em Produto Vetorial

- Calcular o produto vetorial entre o vetor definido pela aresta e pelo vetor que define a reta.
  - $+1$ se o componente-z do produto for positivo.
  - -1 caso contrário.

Regra do Winding-Number não-zero
Processo Baseado em Produto Escalar

- Encontrar um vetor perpendicular ao vetor do segmento de reta $(v_x, v_y) \rightarrow (-v_y, v_x)$ e fazer produto escalar com o vetor da aresta.
  - $+1$ se o produto for positivo.
  - -1 caso contrário.

Preenchimento de Áreas

Preenchimento de Áreas

- A maioria das API's limita o preenchimento de áreas.
  - **Polígonos** porque são descritos por equações lineares.
  - **Polígonos Convexos** porque assim somente duas arestas são cruzadas.
  - Contudo é possível preencher o interior de qualquer tipo de forma utilizando ferramentas de desenho.

Preenchimento de Áreas

Preenchimento de Áreas

- Existem basicamente duas formas:
  1. Determinar os intervalos de preenchimento usando **scanlines** e preencher o interior.
     - Indicado para **polígonos**.
  2. A partir de um ponto, colorir a vizinhança até encontrar as **bordas**.
     - Indicado para formas mais complexas.

Preenchimento de Áreas

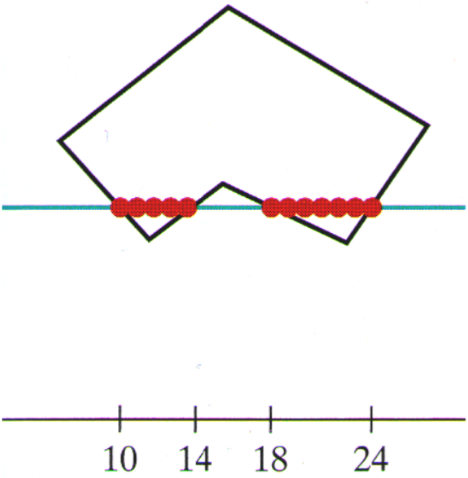
Algoritmo Scanline

- Primeiro determina-se as **intersecções** das *scanlines* com o **polígonos**.
- Então, as **secções** da *scanline* que residirem **dentro** do polígono são **coloridas**.
    - Usa a **regra par-ímpar**.

Para polígonos, 2 equações lineares são utilizadas para encontrar as intersecções.

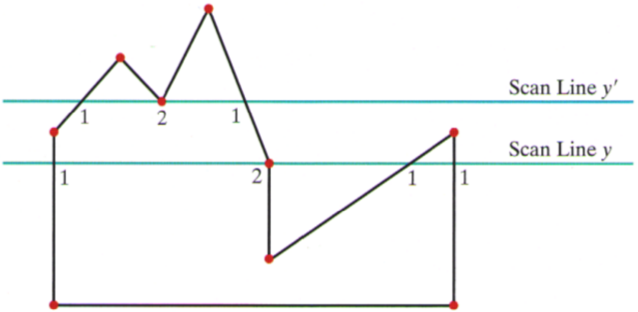
- Calcula-se as intersecções de um polígono da esquerda para a direita.

Preenchimento de Áreas



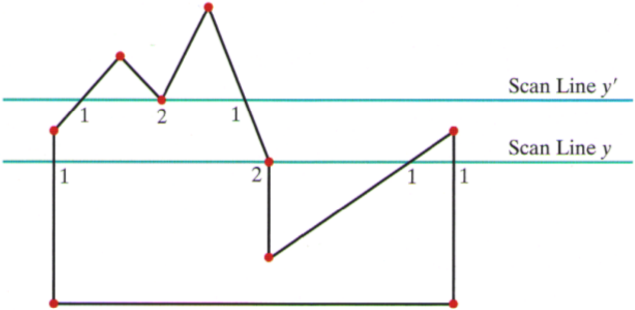
Preenchimento de Áreas

- **Problemas** quando a *scanline* passa por um **vértice**.
  - Intersecta dois polígonos simultaneamente.



Preenchimento de Áreas

- A **contagem** da intersecção deve ser diferente dependendo da **topologia**
  - Duas arestas de **lados opostos** da *scanline*: conta **uma intersecção**.
  - Duas arestas de **mesmo lado** da *scanline*: conta **duas intersecção**



Preenchimento de Áreas

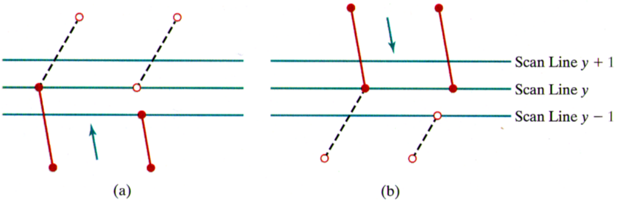
Solução para o Problema da *scanline*

- Para descobrir se as arestas são opostas:
  - Definir a **fronteira** do polígono de forma **anti-horária** (ou horária) e observas as mudanças em y.
  - Se os 3 vértices de duas **arestas consecutivas** são monotonicamente **crescentes** (ou decrescentes) conta somente **uma intersecção**.
  - Caso **contrário duas**.

Preenchimento de Áreas

Solução para o Problema da *scanline* - Pré processada

- Para descobrir se as arestas são opostas:
  - Definir a **fronteira** do polígono de forma **anti-horária** (ou horária) e observas as mudanças em y.
  - Neste caso a **aresta inferior** pode ser **reduzida** para assegurar somente uma intersecção.



Preenchimento de Áreas

Algoritmo *scanline* (intersecção)

- A intersecção da i-ésima *scanline* com uma aresta $(x_1, y_1), (x_2, y_2)$ é calculada com duas equações.

$$
\begin{aligned}
y &= i \\
x &= y/m - b \\
m &= (y_2 - y_1)/(x_2 - x_1) \\
b &= y_1/m - x_1
\end{aligned}
$$

Preenchimento de Áreas

Algoritmo *scanline* (intersecção)

- É possível acelerar esse processo usando um abordagem incremental.

$$m = (y_{k+1} - y_k)/(x_{k+1} - x_k)$$

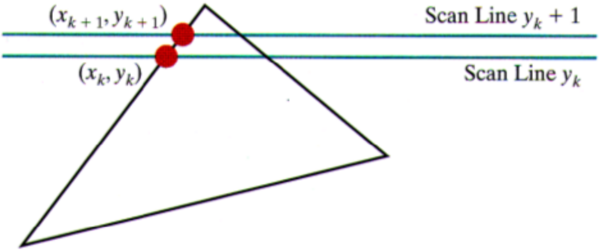
- Como entre duas *scanlines* consecutivas:

$$y_{k+1} - y_k = 1$$

- Então:

$$m = 1/(x_{k+1} - x_k)$$
$$x_{k+1} = x_k + 1/m$$

Preenchimento de Áreas



Preenchimento de Áreas

Algoritmo *scanline* (intersecção)

- Mas ainda é possível utilizar somente inteiros, lembrando que:

$$m = \frac{\Delta y}{\Delta x}$$

- então:

$$x_{k+1} = x_k + \frac{\Delta y}{\Delta x}$$

Preenchimento de Áreas

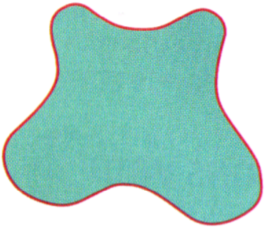
Algoritmo *scanline* (intersecção)

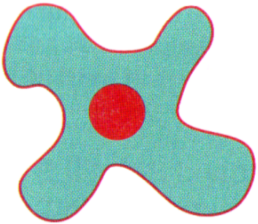
- Para polígonos **convexos**, só existe **um bloco de pixels** subsequente em cada *scanline*.
  - Só processa a *scanline* até encontrar **duas intersecções**.

Preenchimento de Regiões Irregulares

Preenchimento de Regiões Irregulares

- É possível preencher uma região irregular selecionando um pixel e **pintando** os **pixels vizinhos** até alcançar as bordas.



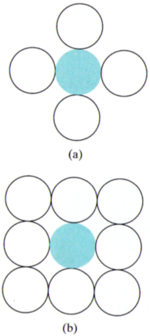


Preenchimento de Regiões Irregulares

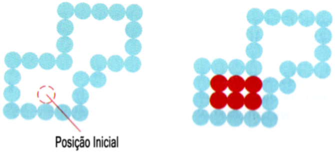
Preenchimento de Regiões Irregulares

- Se a borda de uma região tem a mesma cor, é possível preencher essa região pixel por pixel até atingir a cor da borda.
  - Normalmente usado em programas gráficos.
  - Começa com um ponto inicial (x,y) e testa os vizinhos para ver a cor, se não for borda, preenche.

Preenchimento de Regiões Irregulares



(a) Diferentes testes de vizinhança



(b) Problemas com a máscara de quatro vizinhos

Preenchimento de Regiões Irregulares

```
void fill(int x, int y, int fillColor, int borderColor){

    int color;
    getPixel(x,y,color);

    if ( (color != borderColor) && (color != fillColor) ){
        setPixel(x, y, fillColor);
        fill(x+1, y, fillColor, borderColor);
        fill(x-1, y, fillColor, borderColor);
        fill(x, y+1, fillColor, borderColor);
        fill(x, y-1, fillColor, borderColor);
    }
}
```

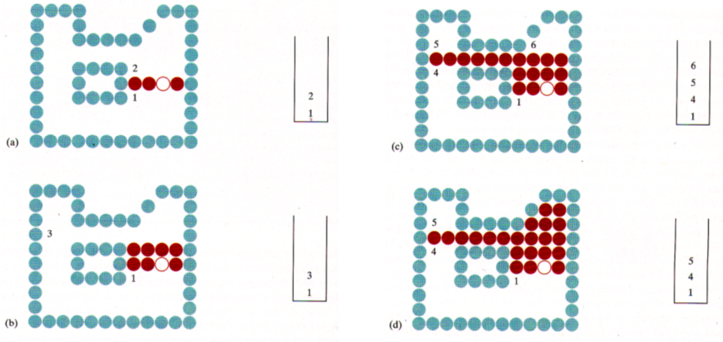
Preenchimento de Regiões Irregulares

Preenchimento de Regiões Irregulares

- **Porblemas** se algum **pixel interior** já for da **cor escolhida** para ser preenchida.
  - Algum ramo da recursão pode ser descartado.

- Pode levar ao consumo excessivo de memória devido a recursão.
  - Uma solução é empilhar, ao invés de pixels vizinhos, blocos de pixels sucessivos (o pixel inicial desses).

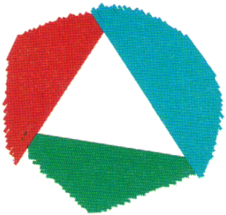
Preenchimento de Regiões Irregulares



Preenchimento de Regiões Irregulares

Algoritmo *Fill-flood*

- As vezes é necessário colorir uma área que **não é definida** apenas por uma **cor de borda**.
  - Ao invés de procurar uma cor de borda, procurar por uma **cor de interior**.
  - Se o interior tem mais de uma cor, pode-se inicialmente substituir esta cor para que todos os pixels do interior tenham a mesma cor.



Preenchimento de Regiões Irregulares

```
void fill(int x, int y, int fillColor, int interiorColor){

    int color;
    getPixel(x,y,color);

    if ( color == interiorColor ){
        setPixel(x, y, fillColor);
        fill(x+1, y, fillColor, interiorColor);
        fill(x-1, y, fillColor, interiorColor);
        fill(x, y+1, fillColor, interiorColor);
        fill(x, y-1, fillColor, interiorColor);
    }
}
```

Preenchimento de Regiões Irregulares

```
void fill(int x, int y, int fillColor, int interiorColor){

    int color;
    getPixel(x,y,color);

    if ( color == interiorColor ){
        setPixel(x, y, fillColor);
        fill(x+1, y, fillColor, interiorColor);
        fill(x-1, y, fillColor, interiorColor);
        fill(x, y+1, fillColor, interiorColor);
        fill(x, y-1, fillColor, interiorColor);
    }
}
```

Preenchimento de Regiões Irregulares

Algoritmo *Fill-flood*

- No caso de preenchimento de áreas irregulares, cada pixel está sendo "pintado" com uma cor.

No caso de rendering de superfícies, cada pixel é pintado com a cor determinada pela aplicação do algoritmo de iluminação + tonalização (shading).