

Aula – Computação Gráfica



Recorte



Slides para uso pessoal e exclusivo durante o período de aula. Distribuição ou qualquer uso fora do escopo da disciplina é expressamente proibido.

1

1

Porque Recortar?

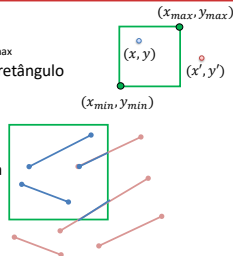
- O processo de renderização é caro
- Ele é aplicado a todos os objetos da cena a todo momento
- É interessante eliminar os objetos que não são de interesse
 - Se o objeto está fora do volume de visualização
 - Pode-se eliminar facilmente
- Alguns objetos estão parcialmente fora
 - Então, devem ser recortados
- O recorte também pode ser aplicada para visualização
 - Ou seja, para ver o objeto recortado
 - Ex. quando deseja-se olhar o interior do objeto

2

2

Recorte de Linhas em 2D

- Recortando os pontos finais
 - Se $x_{\min} < x < x_{\max}$ e $y_{\min} < y < y_{\max}$
 - Então o ponto está dentro do retângulo
- Análise dos pontos finais
 - Se os dois estão dentro
 - Faça o aceite trivial
 - Se um está dentro e outro fora
 - Tem que recortar
 - Se os dois estiverem fora
 - Não se sabe



3

3

Recorte de Linhas em 2D

- Força bruta
 - Resolver as equações da reta para a linha e os quatro lados do retângulo $y = mx + b$
 - Só funciona para linhas infinitas
 - Não trata segmentos de reta
 - Não serve para linhas verticais

4

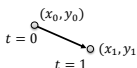
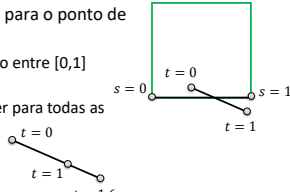
4

Recorte de Linhas em 2D

- Equação paramétrica para segmento de linha

$$X = x_0 + t(x_1 - x_0)$$

$$Y = y_0 + t(y_1 - y_0) \quad 0 \leq t \leq 1$$

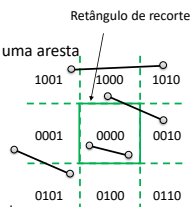
$$P(t) = P_0 + t(P_1 - P_0) = (1 - t)P_0 + t(P_1)$$

- A linha está no retângulo se para o ponto de interseção
 - As variáveis t_{line} e s_{edge} estão entre $[0,1]$ simultaneamente
 - É devagar, pois precisa fazer para todas as arestas

5

5

Recorte de Linhas em 2D

- Algoritmo de Cohen-Sutherland
 - Divida o plano em 9 partes
 - Compute o bit de sinal entre um vértice e uma aresta
 - $Y_{max} = Y; Y = Y_{min}; X_{max} = X; X = X_{min}$
 - O ponto está dentro se todos os bits são 0
 - Código de 4 bits
 - Primeiro bit: acima da aresta do topo
 - Segundo bit: abaixo da aresta de baixo
 - Terceiro bit: a direita da aresta da direita
 - Quarto bit: a esquerda da aresta da esquerda



6

6

Recorte de Linhas em 2D

- Algoritmo de Cohen-Sutherland

- Compute o código para os dois vértices da linha

- OC_0 e OC_1

- Linhas totalmente dentro

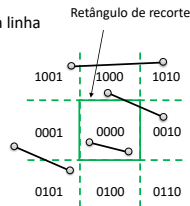
- Isto é, com $OC_0 = 0$ e $OC_1 = 0$

- São trivialmente aceitas

- Linhas totalmente fora

- Isto é, $(OC_0 \text{ AND } OC_1) \neq 0$, com bit comum

- São trivialmente rejeitadas



7

7

Recorte de Linhas em 2D

- Algoritmo de Cohen-Sutherland

- Se não conseguir aceitar ou rejeitar trivialmente

- Use Dividir e Conquistar

- Subdivida a linha em dois segmentos

- Então, aceite ou rejeite trivialmente

- Use o recorte para dividir a linha

- Use o código de bits para escolher as arestas que interceptam

- Para uma dada aresta

- Uma linha intercepta se o bit respectivo é diferente para os vértices

- Escolha uma ordem para checar as arestas (top, bottom, right, left)

- Compute o ponto de interseção

- A aresta de corte fixa X ou Y, que podem ser substituídos na equação

- Itere na nova linha, pois novos cortes podem ocorrer (E-I -> E-H)

8

8

Recorte de Linhas em 2D

- Algoritmo de Cohen-Sutherland

$$\frac{(y - y_0)}{(x - x_0)} = \frac{(y_1 - y_0)}{(x_1 - x_0)}$$

- Monte a equação para x e para y e utilize os valores min e max

```
ComputarOC (x0, y0, oc0);
ComputarOC (x1, y1, oc1);
```

```
repeat
```

```
  Faz rejeite e aceite triviais
```

```
  Pega um ponto de fora do retângulo de recorte
```

```
if TOPO then
```

```
  x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);
```

```
  y = ymax;
```

```
else if BAIXO then
```

```
  x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
```

```
  y = ymin;
```

```
else if DIREITA then
```

```
  y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);
```

```
  x = xmax;
```

```
else if ESQUERDA then
```

```
  y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
```

```
  x = xmin;
```

```
if (x0, y0 é o ponto de fora) then
```

```
  x0 = x; y0 = y; ComputarOC (x0, y0, oc0)
```

```
else
```

```
  x1 = x; y1 = y; ComputarOC (x1, y1, oc1)
```

```
until done
```

9

9

Recorte de Linhas em 3D

- Algoritmo de Cohen-Sutherland
 - Bem parecido com o 2D
 - Divide o volume em 27 regiões
 - Código de 6 bits
 - Fazer a aceitação e rejeição trivial

Planos					
Traseiro	Frontal	Topo	Baixo	Direita	Esquerda
000000	010000	001000	000000	000000	000001
100000	000000	000000	000100	000010	000000

10

10

Recorte de Linhas em 3D

- Algoritmo de Cohen-Sutherland
 - Para o volume canônico

```

xmin = ymin = -1; xmax = ymax = 1;
zmin = -1; zmax = 0;

ComputarOC (x0, y0, x1, y1, z0, oc0);
ComputarOC (x1, y1, z1, oc1);
repeat
  Faz rejeição e aceite trivial
  pega um ponto de fora do cubo de recorte
  if TOPO then
    x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);
    z = z0 + (z1 - z0) * (ymax - y0) / (y1 - y0);
    y = ymax;
  else if BAIXO then
    x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
    z = z0 + (z1 - z0) * (ymin - y0) / (y1 - y0);
    y = ymin;
  else if DIREITA then
    y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);
    z = z0 + (z1 - z0) * (xmax - x0) / (x1 - x0);
    x = xmax;
  else if ESQUERDA then
    y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
    z = z0 + (z1 - z0) * (xmin - x0) / (x1 - x0);
    x = xmin;
  else if FRONTAL then
    x = x0 + (x1 - x0) * (zmax - z0) / (z1 - z0);
    y = y0 + (y1 - y0) * (zmax - z0) / (z1 - z0);
    z = zmax;
  else if TRASEIRO then
    x = x0 + (x1 - x0) * (zmin - z0) / (z1 - z0);
    y = y0 + (y1 - y0) * (zmin - z0) / (z1 - z0);
    z = zmin;
  if (x0, y0, z0 is the outer point) then
    x0 = x; y0 = y; z0 = z;
    ComputarOC (x0, y0, x1, y1, z0, oc0)
  else
    x1 = x; y1 = y; z1 = z;
    ComputarOC (x1, y1, z1, oc1)
until done

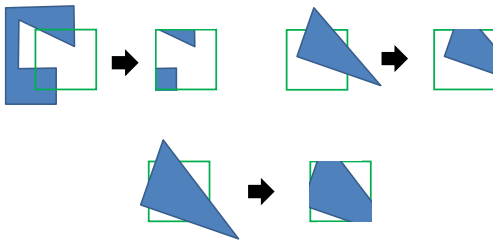
```

11

11

Recorte de Polígonos

- Algoritmo de Sutherland-Hodgman
 - Pode ser generalizado para dimensões maiores

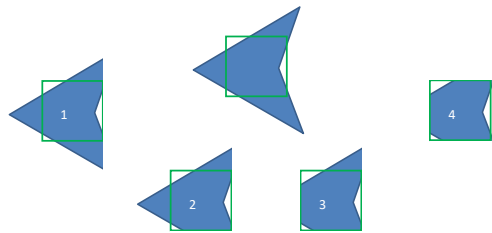


12

12

Recorte de Polígonos

- Algoritmo de Sutherland-Hodgman
 - Pode ser generalizado para dimensões maiores



13

13

Recorte de Polígonos

- Algoritmo de Sutherland-Hodgman
 - Considerando um polígono definido pela lista de vértices *inputList*

```
Point S = inputList.last;
for (Point E in inputList) do
  if (E inside clipEdge) then
    //De fora para dentro
    if (S not inside clipEdge) then
      outputList.add(ComputeIntersection(S,E,clipEdge));
    end if
    outputList.add(E);
  else if (S inside clipEdge) then
    //De dentro para fora
    outputList.add(ComputeIntersection(S,E,clipEdge));
  end if
  S = E;
done
```

14

14

Perguntas ?????

15

15
