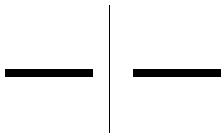
Aula – Computação Gráfica

Rasterização



Drummyfish
https://commons.wikimedia.org/wiki/File:Pixel_vs_subpixel_precision.gif
CC0

Conversão Analítica para Discreta

- Problema
  - Converter valores contínuos para coordenadas discretas
- Por quê isso é um problema?
  - Imagine que queira-se:
    - Desenhar uma reta pintando pontos com coordenadas discretas de um grid, a partir de dois pontos.
  - Por quê isso é difícil?
    - Precisamos definir o que é desenhar em um dispositivo raster
    - Precisamos definir o que é uma linha em um dispositivo raster
    - Precisamos ponderar eficiência e aparência

Conversão Analítica para Discreta

- É o passo final do pipeline de rasterização
- É o processo de pegar os objetos e converter em pixels
- Os frameworks gráficos fazem isso no final do pipeline
- É feito a todo momento e para cada objeto
  - Portanto, deve ser rápido
- Para a renderização 3D também leva em consideração
  - Luz, shading, etc.

- Vamos focar nas primitivas mais simples
  - Exemplo: linha

Conversão de uma Linha

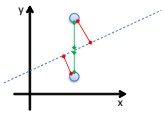
- Casos especiais
  - Horizontal
    - Desenhe o pixel P e incremente a coordenada x de 1
  - Vertical
    - Desenha o pixel P e incremente a coordenada y de 1
  - Diagonal
    - Desenhe o pixel P e incremente as coordenadas x e y de 1
- O que fazer no caso geral?
  - Para uma inclinação < 1
    - Incremente a coordenada x de 1 e escolha o pixel mais perto da linha
    - Outras inclinações são feitas por reflexão
    - Porém, como medimos “mais perto da linha”?

4

4

Conversão de uma Linha

- Por quê podemos usar distâncias verticais?
  - Porque é proporcional a distância verdadeira
  - Portanto, o ponto com a distância vertical menor é o mais próximo
  - Lembrar que a inclinação considerada é <= 1 (primeiro octante)



5

5

Conversão de uma Linha

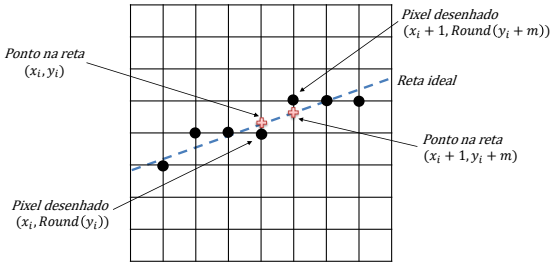
- Algoritmo Básico
  - Ache a equação que conecte dois pontos P e Q
  - Comece com o ponto mais a esquerda
  - Incremente x_i de 1 para calcular $y_i = m*x_i + B$
    - M é a inclinação, B = intercepção em y, e y_i é um float
  - Desenhe o pixel (x_i , Round(y_i)), em que Round $(y_i) = \lfloor .5 + y_i \rfloor$
- Algoritmo incremental
  - Cada iteração requer uma multiplicação de ponto flutuante
    - Modifique o algoritmo para usar um incremento
    - $(y_{i+1} - y_i) = m * (x_{i+1} - x_i)$
    - $y_{i+1} = y_i + m * (x_{i+1} - x_i)$
    - If $\Delta x = x_{i+1} - x_i = 1$, then $y_{i+1} = y_i + m$
  - A cada iteração fazemos um incremento em y para achar o próximo valor, e arredondamos para inteiro

6

6

Conversão de uma Linha

- Algoritmo incremental



7

7

Conversão de uma Linha

- Algoritmo incremental

```
void Line(int x0, int y0, int x1, int y1) {
    float dy = y1 – y0;
    float dx = x1 – x0;
    float m  = dy / dx; // Precisa tartar dx = 0

    int y = y0;
    for (int x = x0; x < x1; ++x) {
        WritePixel( x, Round(y) ); // Arredondamento é ineficiente
        y = y + m;
    }
}
```

8

8

Conversão de uma Linha

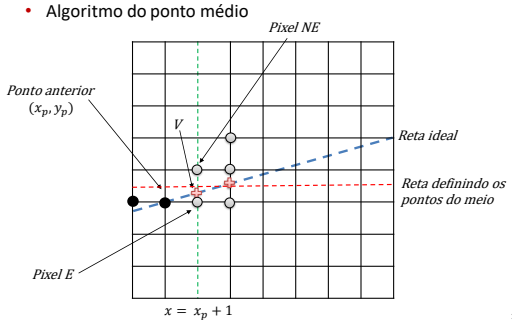
- Algoritmo do ponto médio
  - Assuma que a inclinação da reta é baixa (0 < m < 1)
    - Outras inclinações podem ser tradas por reflexões
  - Chame o ponto mais baixo da esquerda de (x_0, y_0)
  - Chame o ponto mais alto da direita de (x_1, y_1)
  - Assuma que acabamos de selecionar o pixel P em (x_p, y_p)
  - Em seguida, devemos escolher entre
    - O pixel da direita (E pixel), ou o pixel da direita acima (NE pixel)
  - Considere V o ponto de interseção entre a linha de interesse e a reta vertical $x = x_P + 1$

9

9

Conversão de uma Linha

- Algoritmo do ponto médio

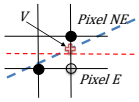


10

10

Conversão de uma Linha

- Algoritmo do ponto médio
  - A linha passa entre E e NE
  - O ponto que está mais perto da interseção *V* deve ser escolhido
  - Observe em que lado da linha o M está:
    - E está mais próximo da linha se o ponto do meio M está acima da linha
    - NE está mais próximo da linha se o ponto do meio M está abaixo da linha
  - A distância entre o pixel escolhido e a linha é sempre menor ou igual a 0.5



11

11

Conversão de uma Linha

- Algoritmo do ponto médio
  - Como verificar quem esta mais próximo?
    - Equação da linha: $f(x) = y = mx + B = \frac{dy}{dx}x + B$
    - Forma implícita: $f(x, y) = ax + by + c = 0$
      - Evita inclinações infinitas
    - Da equação:

$$y\,dx = dy\,x + B\,dx$$
$$dy\,x - y\,dx + B\,dx = 0$$
$$a = dy, b = -dx, c = B\,dx$$

    - Propriedades
      - $f(x_m, y_m) = 0$ quando o ponto *m* está na linha
      - $f(x_m, y_m) < 0$ quando o ponto *m* está acima da linha
      - $f(x_m, y_m) > 0$ quando o ponto *m* está abaixo da linha
      - Nossa decisão será baseada no valor do ponto do meio
        - » $m = M = (x_P + 1, y_P + .5)$

12

12

Conversão de uma Linha

- Algoritmo do ponto médio
  - Varável de decisão *d*
    - Só precisamos do sinal de $f(x_P + 1, y_P + .5)$
  - $d = f(x_P + 1, y_P + .5)$
    - Se $d > 0$ escolha o pixel NE
    - if $d < 0$ escolha o pixel E
    - if $d = 0$ escolha um deles consistentemente
  - Porém, como atualizamos *d* incrementalmente?
    - Baseado na escolha de E ou NE
      - Descubra o M do próximo pixel e o *d* correspondente
    - Podemos derivar o pixel *d* baseado na nossa decisão corrente

13

13

Conversão de uma Linha

- Algoritmo do ponto médio
  - Incrementando *d* se E for escolhido
    - $d_{old} = a(x_P + 1) + b(y_P + .5) + c$
    - $d_{new} = f(x_P + 2, y_P + .5)$
      $= a(x_P + 2) + b(y_P + .5) + c$
    - $d_{new} - d_{old}$ é a diferença incremental ΔE
      - $a(x_P + 2)$ - $a(x_P + 1) = a$
      - $d_{new} = d_{old} + a$, então ΔE = $a = dy$ (2 slides antes)
  - Podemos calcular o próximo *d* incrementalmente
    - $d_{new} = d_{old} + \Delta E = d_{old} + dy$
  - ΔE é o fator de atualização

14

14

Conversão de uma Linha

- Algoritmo do ponto médio
  - Incrementando *d* se NE for escolhido
    - $d_{new} = f(x_P + 2, y_P + 1.5)$
      $= a(x_P + 2) + b(y_P + 1.5) + c$
    - $\Delta NE = d_{new} - d_{old} =$
      $a(x_P + 2) + b(y_P + 1.5)$ - $(a(x_P + 1) + b(y_P + .5))$
      - $d_{new} = d_{old} + a + b$. *Thus* $\Delta NE = a + b = dy - dx$
    - Thus, incrementally,
      $d_{new} = d_{old} + \Delta NE = d_{old} + dy - dx$

15

15

Conversão de uma Linha

- Algoritmo do ponto médio (RESUMO)
  - O primeiro pixel é o primeiro ponto (x_0, y_0)
    - Pode ser calculado diretamente
  - A cada passo escolha entre 2 pixels baseado em *d*
  - Atualize o valor de *d* com o respectivo delta ΔE ou ΔNE

16

16

Conversão de uma Linha

- Algoritmo do ponto médio (Primeiro Ponto)
  - O primeiro pixel é o primeiro ponto (x_0, y_0)
    - $d = d_{start}$ is at $(x_0 + 1, y_0 + .5)$
      - f $(x_0 + 1, y_0 + .5)$

        $= a(x_0 + 1) + b(y_0 + .5) + c$

        $= ax_0 + by_0 + a + \frac{b}{2} + c$

        $= f(x_0, y_0) + a + \frac{b}{2}$
    - Mas (x_0, y_0) é um ponto na linha, então f $(x_0, y_0) = 0$
      - $d_{start} = a + \frac{b}{2} = dy - \frac{dx}{2}$
    - Para eliminar a fração podemos multiplicar f (x, y) por 2
      - Isso não altera o sinal da variável *d*
      - $d_{start} = 2dy - dx$

17

17

Conversão de uma Linha

```
void MidpointLine(int x0, int y0, int x1, int y1) {
    int dx = (x1 - x0), dy = (y1 - y0);
    int d = 2 * dy - dx;
    int incrE = 2 * dy;
    int incrNE = 2 * (dy - dx);
    int x = x0, y = y0;
    WritePixel(x, y);

    while (x < x1) {
        if (d <= 0)
            d = d + incrE;    // Pixel E
        else {                // Pixel NE
            d = d + incrNE; ++y;
        }
        ++x;
        WritePixel(x, y);
    }
}
```

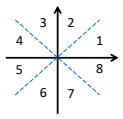
Esse algoritmo é identico ao de Bresenham's, porém derivado de forma diferente

18

18

Conversão de uma Linha

- Algoritmo do ponto médio
  - Octantes
    - O algoritmo só é valido para o primeiro octante
    - Descubra o octante da reta
      - Veja quem é maior entre *dx* e *dy*
      - Veja o sinal de *dx* e *dy*
    - Inverta *x* e *y* apropriadamente
      - Para colocar a reta no primeiro octante
    - Calcule os pontos da reta
    - Inverta o *x* e *y* dos pontos calculados para o octante original



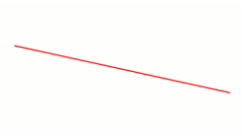
Converte (x, y) para o primeiro octante:
Se estiver no octante 1: returna (x, y)
Se estiver no octante 2: returna (y, x)
Se estiver no octante 3: returna (y, -x)
Se estiver no octante 4: returna (-x, y)
Se estiver no octante 5: returna (-x, -y)
Se estiver no octante 6: returna (-y, -x)
Se estiver no octante 7: returna (-y, x)
Se estiver no octante 8: returna (x, -y)

19

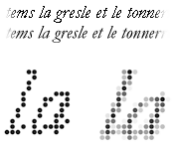
19

Conversão de uma Linha

- Esse algoritmo para linha não lida com antialiasing
- Para obter linhas com antialiasing
  - Use, por exemplo, o algoritmo de Xiaolin Wu´s



Dnttlithmmmm
https://commons.wikimedia.org/wiki/File:LineXiaolinWu.gif
CC BY-SA 4.0



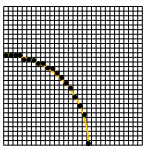
Vierge Marie
https://commons.wikimedia.org/wiki/File:Antialiasing.png
Public Domain

20

20

Conversão de um Círculo

- Versão 1
  - Amostragem não uniforme!
  - Algoritmo
    - For x from $-R$ *to* R
      - $y = \sqrt{R^2 - x^2}$;
      - `WritePixel(`x`, round(`y`))`
      - `WritePixel(`x`, round(`$-y$`));`

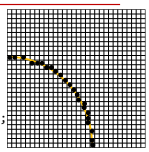


21

21

Conversão de um Círculo

- Versão 2
  - Uniforme, mas sem controle da amostragem!
  - Algoritmo
    - For θ from 0 to 360:
      - `WritePixel(round(`$R\cos(\theta)$`), round(`$R\sin(\theta)$`));`

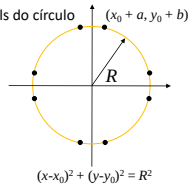


22

22

Conversão de um Círculo

- Versão 3
  - Usa simetria
  - Se $(x_0 + a, y_0 + b)$ está no círculo centrado em (x_0, y_0)
    - Então, $(x_0 \pm a, y_0 \pm b)$ e $(x_0 \pm b, y_0 \pm a)$ também estão
    - Existem 8 simetrias
  - Reduz o problema
    - Em achar 1/8 dos pixels do círculo

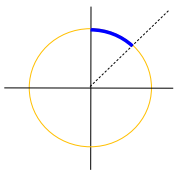


23

23

Conversão de um Círculo

- Versão 3
  - Ache os 1/8 de pixels no topo direito do círculo de raio *R*
  - O círculo começa em $(x_0, y_0 + R)$
  - Vamos usar outro algoritmo incremental com variável de decisão
    - Usando o ponto do meio



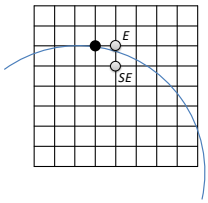
24

24

Conversão de um Círculo

- Versão 3

```
int x = x0, y = y0 + R; WritePixel(x, y);

for (x = x + 1; (x - x0) < (y - y0); x++)
{
    if (decision_var < 0) {
        // Pixel E
        // Atualiza variável de decisão
    } else {
        // Pixel SE
        // Atualiza variável de decisão
        y--;
    }

    WritePixel(x, y);
}
```



25

25

Conversão de um Círculo

- Versão 3
  - O que precisamos para o algoritmo incremental?
    - Negativo se movermos para E, ou positivo se movermos para SE
      - Ou vice versa
    - Seguindo a estratégia da linha
      - Usamos a equação implícita do círculo
      - $f(x,y) = x^2 + y^2 - R^2 = 0$
      - $f(x,y)$ é zero no círculo, negativo dentro e positivo fora
    - Se estamos no pixel (x,y) examine o próximo pixel
      - $(x+1, y)$ e $(x+1, y-1)$
    - Compute *f* para o ponto do meio

26

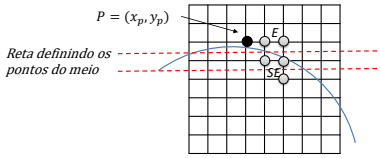
26

Conversão de um Círculo

- Versão 3
  - Avalie $f(x,y) = x^2 + y^2 - R^2$ para o ponto $\left(x+1, y-\frac{1}{2}\right)$
  - Queremos saber o sinal de *f* para:

$$f\left(x+1, y-\frac{1}{2}\right) = (x+1)^2 + (y-\frac{1}{2})^2 - R^2$$

  - Se for negativo, o ponto do meio está dentro, escolha E
  - Se for positivo, o ponto do meio está fora, escolha SE



27

27

Conversão de um Círculo

- Versão 3
  - Como podemos computar a distância vertical em pontos sucessivos?
    $$f(x,y) = d = (x+1)^2 + \left(y - \frac{1}{2}\right)^2 - R^2$$
  - Solução:
    - $f(x+1,y) - f(x,y) = \Delta_E\ (x,y) = 2x + 3$
    - $f(x+1,y-1) - f(x,y) = \Delta_{SE}(x,y) = 2x - 2y + 5$
  - Outro problema:
    - O incremento não é constante
    - Deve ser calculado a todo passo

32

32

Conversão de um Círculo

- Versão 3
  - Solução
    - Fazer a atualização do incremento a cada passo
  - Calcular a diferença entre os deltas para saber como atualizar os deltas

$$\Delta_E(x+1,y) - \Delta_E(x,y) = 2$$
$$\Delta_E(x+1,y-1) - \Delta_E(x,y) = 2$$

$$\Delta_{SE}(x+1,y) - \Delta_{SE}(x,y) = 2$$
$$\Delta_{SE}(x+1,y-1) - \Delta_{SE}(x,y) = 4$$

33

33

Conversão de um Círculo

- Versão 3
  - Para cada passo, compute o novo$\Delta_E(x,y)$ a partir do antigo
  - Faça o mesmo para $\Delta_{SE}(x,y)$
  - Decida a posição do novo pixel $(a+1,b)$ ou $(a\ +\ 1, b\ -\ 1)$
    - Usando o delta apropriado e calculado previamente
  - Resumo:
    - *Look at d to decide which to draw next, update x and y*
    - *Update d using $\Delta_E(a,b)$ or $\Delta_{SE}(a,b)$*
    - *Update each of $\Delta_E(a,b)$ and $\Delta_{SE}(a,b)$ for future use*
    - *Draw pixel*

34

34

Conversão de um Círculo

- Versão 3

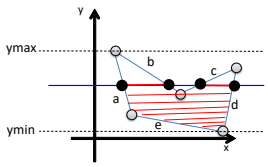
```
MidpointEighthCircle(R) { /* 1/8th de um círculo com raio R */
    int x = 0, y = R;
    int deltaE   = 2 * x + 3;
    int deltaSE  = 2 * (x - y) + 5;
    float decision = (x + 1) * (x + 1) + (y - 0.5) * (y - 0.5) - R*R;

    WritePixel(x, y);
    while ( y > x ) {
        if (decision < 0) { // Pixel E
            x++; WritePixel(x, y);
            decision += deltaE;
            deltaE += 2; deltaSE += 2; // Atualiza os deltas
        } else { // Pixel SE
            y--; x++; WritePixel(x, y);
            decision += deltaSE;
            deltaE += 2; deltaSE += 4; // Atualiza os deltas
        }
    }
}
```

35

35

Rasterização de polígonos

- Algoritmo clássico
  - Ordene as arestas
    - Chave primária: ymin
    - Chave secundária: xmin
    - Ex: (e,d,a,b,c)
  - Varra de ymin até ymax
  - Preencha intervalos horizontais de pares de arestas

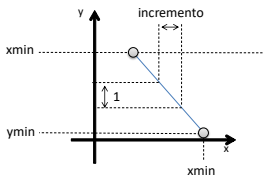


36

36

Rasterização de polígonos

- Algoritmo clássico
  - Informação das arestas
    - Y inicial (mínimo)
    - Y final (máximo)
    - X corrente
    - Incremento em X, dx
  - Lista de arestas
    - Ordenadas pelo y e x iniciais
  - Lista de arestas ativas
    - Arestas tocando a linha de varredura corrente
    - Ordenadas pela coordenada x que intercepta a linha de varredura

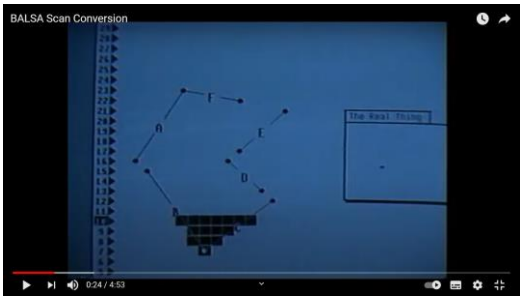


37

37

Rasterização de polígonos

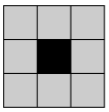
Balsa Video - http://www.youtube.com/watch?v=GXi32vnA-2A



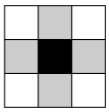
38

Preenchimento (Flood Fill)

- Não é um algoritmo de rasterização
  - Ocorre na imagem e não faz conversão de analítico para discreto
- Preenche uma região baseado na conectividade dos pixels
  - 4-connected
  - 8-connected



Master Ugly
https://commons.wikimedia.org/wiki/File:Sasiedztwa_4_8.svg
Public Domain



André Karwath aka Aka
https://commons.wikimedia.org/wiki/File:Recursive_Flood_Fill_4_(aka).gif
https://commons.wikimedia.org/wiki/File:Recursive_Flood_Fill_8_(aka).gif
CC BY-SA 2.5

40

40

Preenchimento (Flood Fill)

- Flood-fill (pixel, cor-alvo, nova-cor):
  - Se a cor do pixel for igual a nova-cor
    - Retorna sem fazer nada
  - Se a cor do pixel for diferente da cor-alvo
    - Retorna sem fazer nada
  - Atualiza a cor do pixel com a nova-cor
  - Visita os pixels vizinhos (4 ou 8 vizinhos)
    - Flood-fill (pixel do sul, cor-alvo, nova-cor)
    - Flood-fill (pixel do norte, cor-alvo, nova-cor)
    - Flood-fill (pixel do leste, cor-alvo, nova-cor)
    - Flood-fill (pixel do oeste, cor-alvo, nova-cor)

41

41

Perguntas ?????

42