Exercícios Revisão 01 - Computação Gráfica

Gustavo Lopes Rodrigues

9 de setembro de 2021

Transformações Geométricas

1) Coordenadas homogêneas permite o tratamento algébrico de pontos no infinito.

2) Escala

$$P' = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \times \begin{bmatrix} P_x \\ P_y \end{bmatrix} \rightarrow P' = \begin{bmatrix} S_x * P_x \\ S_y * P_y \end{bmatrix}$$

Rotação

$$P' = \begin{bmatrix} cos\ \alpha & -sen\ \alpha \\ sen\ \alpha & cos\ \alpha \end{bmatrix} \times \begin{bmatrix} P_x \\ P_y \end{bmatrix} \rightarrow P' = \begin{bmatrix} P_x * cos\ \alpha - P_y * sen\ \alpha \\ P_x * sen\ \alpha - P_y * cos\ \alpha \end{bmatrix}$$

$$P_x * S_x = P_x * cos\ \alpha - P_y * sen\ \alpha \tag{1}$$

$$S_x = cos\ \alpha - P_y * sen\ \alpha$$

$$P_y * S_y = P_x * sen\ \alpha - P_y * cos\ \alpha \tag{2}$$

$$S_y = P_x * sen\ \alpha - cos\ \alpha$$

3) Essa movimentação pode ser evitada, garantindo que todo objeto tenha um ponto central, dessa forma, para fazer as operações, o vetor para escalonar é a diferença do vértice e esse ponto central.

4) O processo de multiplicação das matrizes de transformação requer que a ordem das operações seja mantida(da última para a primeira operação), pois garante que as operações sejam preservadas, de acordo com a equação geral das transformações 2D

$$[P'] * [M_t] * [P] \tag{3}$$

5)
   a. A(-1,-3) → A'(-2,2);
   B(-2,8) → B'(-3,13);
   C(9,2) → C' (8,7);

   b. A(-1,-3) → A'(-2.36,-2.09);
   B(-2,8) → B'(2.26,7.92);
   C(9,2) → C' (8.79,-2.76);

   c. A(-1,-3) → A'(-2.09,-2.36);
   B(-2,8) → B'(-2,8);
   C(9,2) → C' (2.76,8.79);

d. A(-1,-3) $\rightarrow$ A'(-0.5,-6);
B(-2,8) $\rightarrow$ B'(-1,24);
C(9,2) $\rightarrow$ C' (4.5,21);

e. A(-1,-3) $\rightarrow$ A'(1,-3);
B(-2,8) $\rightarrow$ B'(2,8);
C(9,2) $\rightarrow$ C' (-9,2);

Rasterização de Retas

6. O valor de delta refere-se ao dx e dy, ou seja, a diferença entre ao par (x,y) do ponto inicial e final da reta. Logo, para atravessar o canvas e preencher os pontos da reta, o número de iterações será essa diferença.

7. Para o algoritmo do DDA, o motivo de usar apenas valores positivos se dá no momento em que o passo de **x** precisa ser incrementado. Se não usar o valor absoluto nessa equação não será possível desenhar o deslocamento da linha AB, contudo se todo o algoritmo for modificado para usar um passo de **x** negativo, será possível sim usar valores de delta negativos. Já no caso de Bresenham, é impossível, pois seu algoritmo é bem mais complexo e divide o ângulo da reta em 8 partes, valores negativos implicariam em mudar consideravelmente sua fórmula, uma vez que para evitar deltas negativos no algoritmo de Bresenham, define que deve-se inverter a direção do desenho para que volte a ser usado apenas valores positivos.

DDA

8. Os valores são arredondados apenas na visualização, pois o posicionamento dos pixeis são representados por número inteiros. Como o algoritmo DDA trabalha com pontos flutuantes, precisamos arrendondar esses valores na hora de exibí-los no canvas

9. Durante todo o algoritmo do DDA, é importante manter os valores em ponto flutuante para manter-se a maior precisão possível do formato da linha que será desenhada na tela. A transformação para número inteiro se dá por conta dos displays que usamos serem construídos apenas com números inteiros de pixels. Não sendo possível desenhar em uma tela em uma posição fracionada.

10. a. AB – A(-1,4) e B(5, 7)
-1, 4
0, 5
1, 5
2, 6
3, 6
4, 7
5, 7

b. BA – B(5, 7) e A(-1, 4)
5, 7
4, 7
3, 6
2, 6
1, 5
0, 5
-1, 4

c. CD – C(-1, 4) e D(3, 8)
   -1, 4
   0, 5
   1, 6
   2, 7
   3, 8

d. EF – E(2, 0) e F(6, 0)
   2, 0
   3, 0
   4, 0
   5, 0
   6, 0

e. GH – G(1, 3) e H(1, 6)
   1, 3
   1, 4
   1, 5
   1, 6

Bresenham

11. O algoritmo de bresenham trabalha com inteiros em vez de pontos flutuantes(floats), isso permite com que as linhas sejam mais precisas, quando comparadas ao algoritmo DDA. Sem contar o gasto em memória é menor, já que precisamos de menor bytes para armazenar valores inteiros do que pontos flutuantes.

12. O algoritmo de Bresenham se torna extremamente eficiente ao dividir os ângulos de desenho de reta em 8 partes. O uso apenas de valores positivos se dá por conta do passo do algoritmo. Para se desenhar uma linha deve-se ir desenhando da esquerda para a direita. O inverso implicaria em modificar drasticamente a implementação do algoritmo para permitir que linhas possam ser desenhadas da direita para a esquerda, o que só aumentaria a complexidade desse, uma vez que apenas invertendo os pontos resultaria na mesma linha. Por exemplo, se um ponto A está a direita de B, não é necessário implementar a escrita da linha da direita para a esquerda e sim trocar a ordem dos pontos, desenhando de B para A.

13. A ideia do algoritmo em sempre atualizar **x** antes da variável de controle **p** é em sempre começar desenhando naquele pixel para depois ver se será necessário desenhar em um pixel acima ou abaixo desse, essa foi uma escolha de implementação feita por Bresenham. Se a linha que estiver sendo desenhada estiver o **m** maior que 1, então teremos que atualizar **y** ao invés de **x**. E seguir a mesma ideia de posteriormente atualizar a variável **p**, para ver se precisamos desenhar em outro pixel em **x**.

14. Quando a variável **p** for positiva, incrementamos o valor do **y**, isso acontece pois **p** é a variável de decisão que irá conduzir ao algoritmo qual pixel ele deve tomar na hora de formar a linha. Como **p** é positivo, significa que a diferença entre d1 e d2(o pixel anterior e posterior) é maior que zero, por isso, precisamos que o valor de y incremente.

15. a. AB – A(-1,4) e B(5, 7)
   -1, 4
   0, 5
   1, 5
   2, 6
   3, 6

4, 7
5, 7

b. BA – B(5, 7) e A(-1, 4)
5, 7
4, 6
3, 6
2, 5
1, 5
0, 4
-1, 4

c. CD – C(-1, 4) e D(3, 8)
-1, 4
0, 5
1, 6
2, 7
3, 8

d. EF – E(2, 0) e F(6, 0)
2, 0
3, 0
4, 0
5, 0
6, 0

e. GH – G(1, 3) e H(1, 6)
1, 3
1, 4
1, 5
1, 6

Rasterização de Circunferências

16. O algoritmo projeta um circulo perfeito, logo, ele só precisa calcular apenas um octante, e o resto, o algoritmo projeta o mesmo resultado para os outros octantes.

17. O que restringe o desenho da circunferência no segundo quadrante é as linhas: **desenharPixel(xc -x, yc - y)** e **desenharPixel(xc - y, yc - x)**. Todas as operações de escrita do pixel quando o x e o y são negativos.

18. A atualização do 'x' precisa ser atualizado antes da variável 'p' e 'y', pois o cálculo da variável de decisão 'p' requer o uso do valor em 'x', que precisa ser atualizado. Enquanto que o 'y', apenas atualiza, se 'p' for maior que zero, justificando a atualização de 'x' anterior as outras duas variáveis.

19. Durante chamada da função de desenhar os 8 pontos do círculos as posições do X e do Y que não estão na origem são utilizados. Dessa forma, durante o algoritmo em si não é necessário saber se o circulo não está na origem, só quando os pontos serão desenhados.

20. a. x = 2, y = 4, p = 5
Desenhar círculo:
xc + x = 1 , yc + y = 6

xc - x = -3 , yc + y = 6
xc + x = 1 , yc - y = -2
xc - x = -3 , yc - y = -2
xc + x = 1 , yc + y = 6
xc + x = 3 , yc + y = 4
xc - x = -5 , yc + y = 4
xc + x = 3 , yc - y = 0
xc - x = -5 , yc - y = 0

b. xc = -1, yc = 2, r = 5
x = 3, y = 3, p = 15
Desenhando círculo:
xc + x = 3 , yc + y = 3
xc - x = -3 , yc + y = 3
xc + x = 3 , yc - y = -3
xc - x = -3 , yc - y = -3
xc + x = 3 , yc + y = 3
xc + x = 3 , yc + y = 3
xc - x = -3 , yc + y = 3
xc + x = 3 , yc - y = -3
xc - x = -3 , yc - y = -3

c. xc = 3, yc = 4, r = 6
x = 2, y = 5, p = -1
Desenhar círculo:
xc + x = 5 , yc + y = 9
xc - x = 1 , yc + y = 9
xc + x = 5 , yc - y = -1
xc - x = 1 , yc - y = -1
xc + x = 5 , yc + y = 9
xc + x = 8 , yc + y = 6
xc - x = -2 , yc + y = 6
xc + x = 8 , yc - y = 2
xc - x = -2 , yc - y = 2

Recorte

21. A ordem dos recortes não altera o resultado, uma vez que a posição que será recortada está salva em um bit que identifica seu lado.

Cohen-Sutherland

22. Os códigos 3 e 7, não são considerados, pois em representação binária seriam 11 e 111 e o primeiro bit representa um lado do recorte o segundo bit o outro lado – o terceiro bit representa que está em baixo -. Dessa forma, é impossível que eles estejam juntos, não é possível que um ponto de uma linha esteja ao mesmo tempo a direita e a esquerda da janela de clipping.

23. Há duas condições de paradas para o algoritmo, no primeiro caso, ambos os pontos estão dentro da área de clipping, no outro caso, ambos os pontos estão fora da área;

24. Uma das condições de parada é quando a linha não está desenhada dentro da área de recorte. Outra razão de parada é se a linha estiver inteiramente dentro da área de clipping. Por fim, as outras razões de parada estão relacionadas à encontrar o ponto de intersecção entre a linha e a janela de recorte, depois de descobrir esse ponto e executar novamente para descobrir o outro ponto (se já estiver dentro já podemos parar), então o algoritmo.

25. c1 e c2 são as váriaveis que vão computar a posição dos pontos da reta inserida no algoritmo. Ambas são números onde cada bit representa a direção que o ponto está, em correspondência a área de clipping. O resultado dessas váriaveis serão armazenados e então é feito a operação descrita "c1 & c2 != 0", que irá fazer um bitwise and. Qualquer número diferente de 0 indicará que a computação de c1 e c2 resultou em um número que está fora da área de clipping.

26. Para os valores originais da cena antes da execução do código, se não forem salvos pelo programador antes de executar o algoritmo, serão todos perdidos.

27. Primeira Linha(AB) → (-1,-3) → (-2,-8)

    Linha calculada:

    x1: -1 ,x2: -2 ,y1: -3 ,y2: 8

    Linha rejeitada

    Segunda linha (BC) → (-2,-8) → (9,2)

    Linha calculada:

    x1: -2 ,x2: 9 ,y1: 8 ,y2: 2

    Linha calculada:

    x1: 1 ,x2: 9 ,y1: 6 ,y2: 2

    Linha aceita:

    x1: 1 ,x2: 6 ,y1: 6 ,y2: 3

    Segunda linha (AC) → (-1,-3) → (9,2)

    Linha calculada:

    x1: -1 ,x2: 9 ,y1: -3 ,y2: 2

    Linha calculada:

    x1: 5 ,x2: 9 ,y1: 0 ,y2: 2

    Linha aceita:

    x1: 5 ,x2: 6 ,y1: 0 ,y2: 0

Liang-Barsky

28. Apenas no final é feito a atualização, pois primeiro é preciso verificar se a reta está dentro da área de clipping, mas também é preciso verificar qual é o ajuste é necessário para que a reta fique dentro da área desejada

29. As estruturas condicionais são aninhadas, pois o programa verifica pelos cantos da área de clipping, ou seja, os cantos direito e esquerdo, tanto inferior e superior.

30. u1 e u2 são parámetros de interseção da linha, quando u1 é maior que u2, rejeitamos essa linha, pois significa que ela não está mais dentro da área de clipping, logo, iniciamos u1 com 0 e u2 com 1, pois $0 < 1$.

31. Primeira Linha(AB) → (-1,-3) → (-2,-8)

    dy = 11.0

    Linha rejeitada

Segunda linha (BC) → (-2,-8) → (9,2)

dy = -6.0 t2 = 0.72 t1 = 0.33

Linha aceita:

x1: 1 ,x2: 6 ,y1: 6 ,y2: 3

Segunda linha (AC) → (-1,-3) → (9,2)

dy =5.0 t2 = 0.7 t1 = 0.6

Linha aceita:

x1: 5 ,x2: 6 ,y1: 0 ,y2: 0

Sutherland-Hodgeman

32) O algoritmo Sutherland – Hodgman funciona estendendo cada linha do polígono do clipe convexo por vez, e selecionando apenas os vértices do polígono que estão visíveis.

33) Para a lista de vértices ser atualizada precisamos considerar 4 possibilidades. A primeira acontece se ambos os vértices estão dentro da área de clipping, a segunda seria se o primeiro vértice está dentro e o segundo vértice está fora. A próxima é quando o primeiro vértice está fora e o segundo dentro e, por último, se ambos estão fora.

34)
- (-1, 1)
- (-1, 6)
- (1, 6)
- (5, 4)
- (5, 1)