Universidade Federal de Santa Maria
Departamento de Computação Aplicada
Prof. Cesar Tadeu Pozzer
Disciplina: Computação Gráfica
pozzer@inf.ufsm.br
18/06/2020

Computação Gráfica 2D

1 Transformações 2D Afins

Uma transformação afim mantém o **paralelismo** de retas. São exemplos a translação, escala, rotação, reflexão e cisalhamento. A projeção em perspectiva não é uma transformação afim. O mesmo vale para transformações de *tapering*, *twist* e *bend* [4].

Transformações afins envolvendo rotações, translações e reflexões preservam ângulos e comprimentos, da mesma forma como linhas paralelas. Para estas três transformações, comprimentos e ângulos entre duas linhas permanecem o mesmo após a transformação [2].

Utiliza-se matrizes para representar transformações. Ver seção de Matrizes.

Translação

Definida pela soma de fatores de translação t_x e t_y as coordenadas x e y do ponto. É uma soma de matrizes

$x' = x + T_x$
$y' = y + T_y$

$P' = P + T$

$$P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad T = \begin{bmatrix} T_x \\ T_y \end{bmatrix} \qquad P' = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \end{bmatrix} = \begin{bmatrix} x + T_x \\ y + T_y \end{bmatrix}$$

Escala

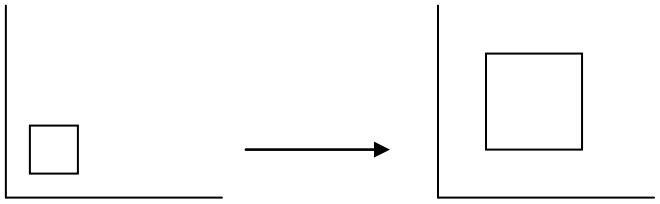
Definida pela multiplicação de fatores de escala S_x e S_y as coordenadas do ponto. É representada por uma multiplicação de matrizes.
$x' = x * S_x$
$y' = y * S_y$

$$P' = T \cdot P$$

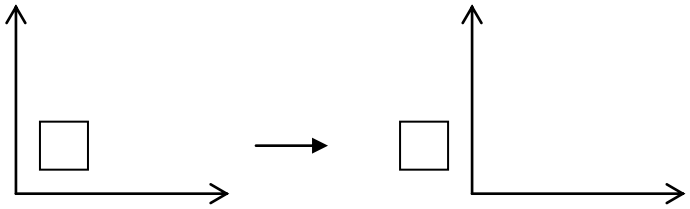
$$P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad T = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \qquad P' = \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} xS_x \\ yS_y \end{bmatrix}$$

Se $S_x > 1$ → aumenta o tamanho do objeto
$S_x < 1$ → diminui o tamanho do objeto
$S_x = 1$ → mantém o tamanho
$S_x = S_y = n$ → escala uniforme

OBS: a escala pode mudar a posição do objeto. Depende do referencial.



Reflexão



$x' = -x$
$y' = y$

$$P' = T \cdot P$$

$$P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad T = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \quad P' = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -x \\ y \end{bmatrix}$$

Cisalhamento

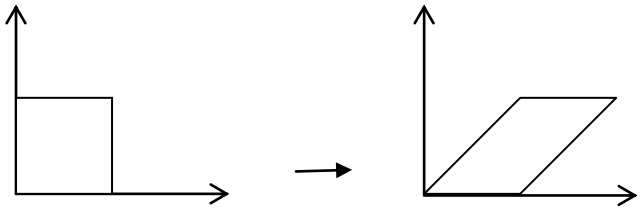
Transformação que distorce o formato do objeto. Aplica-se um deslocamento aos valores das coordenadas x ou y do objeto proporcional ao valor das outras coordenadas de cada ponto transformado.

$x' = x + Sy*y$
$y' = y$

$$P = \begin{bmatrix} x \\ y \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \end{bmatrix} \quad T = \begin{bmatrix} 1 & S_y \\ 0 & 1 \end{bmatrix}$$

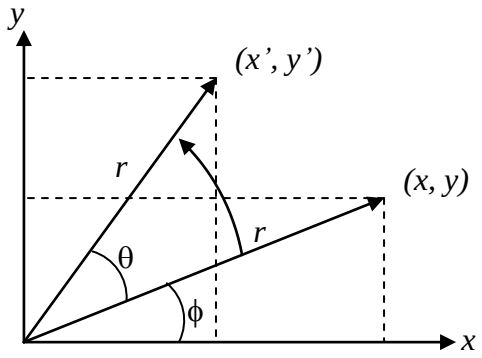
$$P' = T \cdot P$$

$$P' = \begin{bmatrix} 1 & S_y \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + S_y y \\ y \end{bmatrix}$$



Rotação

Rotacionar um vetor significa mudar sua direção segundo algum eixo de rotação. Para um vetor bidimensional, ou seja, paralelo ao plano *xy*, significa reposicioná-lo sobre um caminho circular, como ocorre com o movimento dos ponteiros de um relógio. O eixo de rotação é um vetor perpendicular ao plano *xy* e passa pelo centro de rotação. A rotação é especificada por um ângulo θ (**Coordenadas Polares**) Valores positivos de θ geram uma rotação no sentido anti-horário. Neste exemplo, considera-se que o centro de rotação é a origem do sistema de coordenadas.



Rotação de um vetor da posição *(x,y)* para a posição *(x', y')* segundo um ângulo θ relativo à origem do sistema de coordenadas.

Em uma rotação, a norma *r* do vetor permanece inalterada. Usando identidades trigonométricas, pode-se expressar as coordenadas transformadas em termo dos ângulos ϕ e θ.

$$x' = r\cos(\phi+\theta) = r\cos\phi\cos\theta - r\sin\phi\sin\theta$$
$$y' = r\sin(\phi+\theta) = r\cos\phi\sin\theta + r\sin\phi\cos\theta$$

As coordenadas iniciais do vetor em **coordenadas polares** são

$$x = r\cos\phi$$
$$y = r\sin\phi$$

Substituindo na expressão anterior temos

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

Observação importante: cuidado para não usar *x'* ao calcular *y'*.

Usando-se uma notação matricial, tem-se

$$P' = RP$$
$$P' = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Vetor Perpendicular

Dado um vetor *P*, para encontrar um vetor *Q*, perpendicular a *P* (que forma um ângulo de 90º em sentido anti-horário), usa-se a seguinte fórmula:

$$Q = \langle -P_y \quad P_x \rangle$$

Esta fórmula pode ser demonstrada utilizando-se a notação matricial genérica para rotações sobre um eixo:

$$Q = \begin{bmatrix} \cos(90) & -\sin(90) \\ \sin(90) & \cos(90) \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix}$$

$$Q = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \end{bmatrix} = \begin{bmatrix} 0P_x - 1P_y & 1P_x + 0P_y \end{bmatrix} = \begin{bmatrix} -P_y & P_x \end{bmatrix}$$

Deve-se observar que existem dois vetores perpendiculares em um espaço 2D. Para um vetor em três dimensões, a determinação de um vetor perpendicular é mais vaga. Podem existir **infinitos vetores perpendiculares** (qualquer vetor no plano perpendicular ao vetor P). Em um plano 3D existem 2 vetores perpendiculares.

Coordenadas Homogêneas

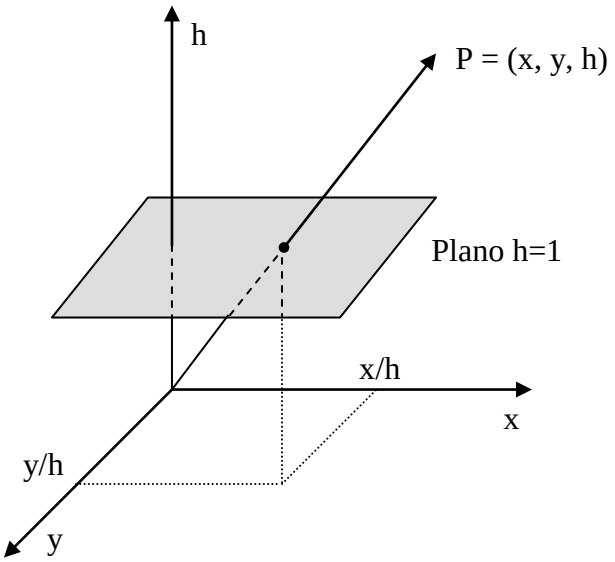
Todas as transformações apresentadas, exceto a translação são representadas como multiplicações de matrizes. Como será visto na seqüência, é comum fazer a combinações de várias transformações em uma única matriz. Como a translação está presente em quase todas as transformações, é fundamental que ela possa ser representada por meio de multiplicação de matrizes. Para isso introduziu-se o conceito de coordenadas homogêneas.

Consiste em adicionar mais uma dimensão à matriz. Cada ponto 2D é representado por *(x,y,h)*. Caso o valor de h não seja 1, deve-se dividir cada componente do vetor por *h*. Assim, o ponto (2, 3, 6) e (4, 6, 12) representam a mesma informação

$$\begin{pmatrix} x & y & h \end{pmatrix} = \begin{pmatrix} \frac{x}{h} & \frac{y}{h} & 1 \end{pmatrix}$$

Este processo é chamado de homogeneização. Neste caso, *x/h* e *y/h* são as coordenadas cartesianas do ponto homogêneo.

A terceira coordenada do ponto pode ser interpretada geometricamente como um plano *h=1* no R^3, como mostrado na figura. O ponto *P'=(x/h, y/h)* é a projeção do ponto *P(x, y, h)* no plano *h=1*. Desta forma, qualquer múltiplo do ponto P homogêneo representa o mesmo ponto.



Fazendo uso de coordenadas homogêneas, a **translação** é definida como

P' = TP

$$P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad T = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \qquad P' = TP = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + T_x \\ y + T_y \\ 1 \end{bmatrix}$$

Escala:

$P' = S \cdot P$

$$P = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad P' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \quad S = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad P' = SP = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} xS_x \\ yS_y \\ 1 \end{bmatrix}$$

Rotação:

$$P' = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \\ 1 \end{bmatrix}$$

Matrizes em Transformações Geométricas

Matriz Ortogonal

Uma matriz quadrada ***A*** é **ortogonal** se $MM^T = \boldsymbol{I}$. Portanto, se ***M*** é ortogonal então

$M^{-1} = M^T$

Uma outra conseqüência é que se ***M*** é ortogonal então *det* ***M*** é igual a 1 ou -1.

Propriedades:
- Se ***A*** e ***B*** são ortogonais então ***C*** = ***AB*** também é ortogonal.
- Se ***A*** é ortogonal então $\boldsymbol{A}^{-1}$ também é ortogonal.

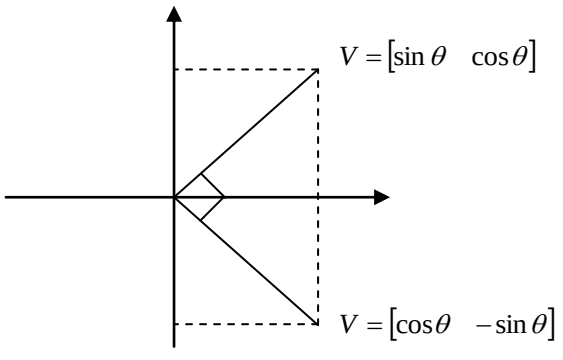
Uma matriz é dita é dita **ortonormal** se o comprimento de cada linha ou coluna, tomados como vetores, tem comprimento 1 (normalizado). Esses vetores também são mutuamente **ortogonais** (formam um ângulo de 90^o cada (perpendiculares) – ver definição de produto interno/escalar).

Teorema: Se M é uma **matriz ortogonal**, então M preserva comprimentos e ângulos. [1]

Uma vez que matrizes **ortonormais** preservam ângulos e comprimentos, elas preservam a estrutura geral do sistema de coordenadas. Deste modo, podem apenas representar combinações de **rotações** e **reflexões**. Matrizes de translação não são ortonormais.

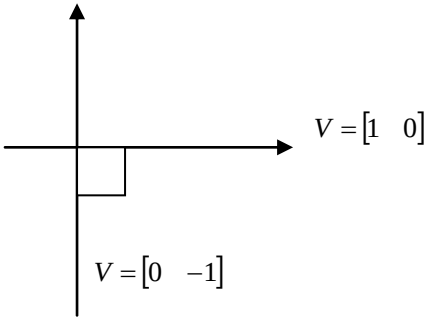
$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$V = \begin{bmatrix} \cos\theta & -\sin\theta \end{bmatrix}, \quad \theta = 45^0$



$$\|V\| = \sqrt{\cos\theta^2 + (-\sin\theta)^2}$$
$$\|V\| = \sqrt{0.707^2 + (-0.707)^2}$$
$$\|V\| = \sqrt{0.5 + 0.5} = \sqrt{1} = 1$$

$$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



$$R = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad R^{-1} = R^T = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Combinações de Transformações

Quando se deseja aplicar várias transformações a vários pontos, ao invés aplicar várias transformações a cada ponto, pode-se combinar todas as transformações em uma única matriz e então aplicar esta matriz resultante aos pontos do objeto, reduzindo-se assim o custo computacional.

Este processo é chamado de concatenação de matrizes, e é executado multiplicando-se as matrizes que representam cada transformação.

OBS: a multiplicação de matrizes não é comutativa, porém é associativa.
AB≠BA (não comutativa)
A(BC) = (AB)C (associativa)
$(FG)^T = G^T F^T$

OBS: Deve-se observar a terceira propriedade: $(FG)^T = G^T F^T$. Ou seja, a forma como a multiplicação é aplicada indica na ordem que os elementos devem estar dispostos.

A seguinte matriz representa a concatenação de uma **translação seguida de uma rotação**. A rotação deve vir antes na multiplicação.

$P' = RTP$ (**Concatenação Correta**)

$$P' = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} \cos\theta & -\sin\theta & T_x\cos\theta - T_y\sin\theta \\ \sin\theta & \cos\theta & T_x\sin\theta + T_y\cos\theta \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$P' = TRP$ (**errado**)

$$P' = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} \cos\theta & -\sin\theta & T_x \\ \sin\theta & \cos\theta & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Observa-se que a transformação TRP≠RTP e que $(FG)^T = G^T F^T$.

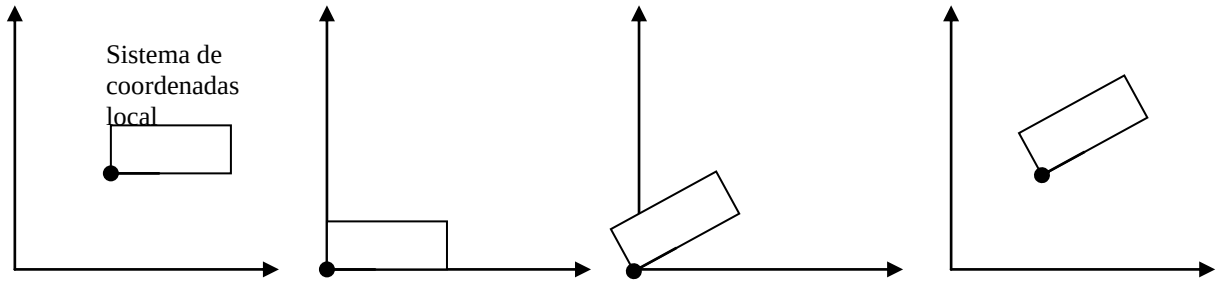
Considerando-se P como P^T, R como R^T e T como T^T, temos:

$$P' = P^T T^T R^T$$

$$P' = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ T_x\cos\theta - T_y\sin\theta & T_x\sin\theta + T_y\cos\theta & 1 \end{bmatrix}$$

No seguinte exemplo deseja-se aplicar uma rotação sobre o eixo do retângulo, ou seja, uma rotação no sistema de coordenadas local do objeto. Para isso, deve-se fazer uma mudança de sistema, ou seja, levar a origem (**pivô**) do sistema de coordenadas do objeto para a origem do sistema de coordenadas do mundo, aplicar a rotação, e transladar o objeto para seu sistema de coordenadas.



Neste tipo de aplicação, é necessário fazer transformações diretas e inversas, ou seja, deve-se aplicar uma matriz de transformação T e em seguida a sua inversa T^{-1}. O cálculo de matrizes inversas é complexo, porém é fácil determinar matrizes inversas de transformações geométricas.

No caso da rotação, por ser uma **matriz ortonormal**, sua inversa de $R(\theta)$ é a transposta (o mesmo vale para matrizes de reflexão), o que é equivalente a $R(-\theta)$.

No caso de uma matriz de translação $T = [T_x \;\; T_y \;\; 1]^T$, T^{-1} é dada por $T^{-1} = [-T_x \;\; -T_y \;\; 1]^T$.

Para uma matriz de escala $S = [S_x \;\; S_y \;\; 1]^T$, $S^{-1} = [1/S_x \;\; 1/S_y \;\; 1]^T$

Assim, para o exemplo anterior temos a seguinte matriz M que representa a concatenação de matrizes de transformação:

$M = T^{1}RT$

$$M = \begin{bmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -T_x \\ 0 & 1 & -T_y \\ 0 & 0 & 1 \end{bmatrix}$$

onde T_x e T_y representam as coordenadas do sistema de coordenadas do objeto em relação ao sistema de coordenadas do mundo.

No caso de rotações sucessivas em um mesmo eixo

$$P' = R(\theta_1)R(\theta_2)P$$

Por serem aditivas, a composição das duas rotações pode ser dada por

$$P' = R(\theta_1 + \theta_2)P$$

Outros exemplos:

- Escala sem alterar a posição do objeto
- Rotação em um eixo arbitrário
- Rotações consecutivas no mesmo eixo
- Rotações consecutivas em diferentes eixos

Implementação de Transformações

Como ilustrado no exemplo anterior, em muitas situações é necessário aplicar (concatenar) várias transformações, incluindo rotação, escala e translações. Neste caso, para criar a matriz de rotação final M deve-se multiplicar várias matrizes.

Antes de partir para uma implementação direta, deve-se observar como a API OpenGL faz para trabalhar com matrizes de transformação. Ele define uma única matriz "global", que é atualizada a cada chamada de comandos glRotate(), glTranslate(), glScale(). A matriz corrente é então multiplicada pela nova matriz correspondente a transformação chamada. O seguinte código em C++ ilustra como criar uma classe para gerenciar a matriz de transformação.

```
Class Matrix
{
    float m[3][3]; //em coordenadas homogêneas
public:
    Matrix();
    void operator *(Matrix m);  //multiplica por outra matriz
    void loadIdentity();        //carrega matriz identidade
    void rotate(float ang);     //concatena matriz de rotação
    void translate(float dx, float dy); //concatena matriz de translação
    void scale(float sx, float sy);     //concatena matriz de escala
    Vector operator*(Vector v);         //multiplica um ponto v pela matriz
    void push(); //empilha matriz (duplica o valor do topo)
    void pop();  //desempilha matriz (remove o topo da pilha)
}
```

Na prática, ao invés de ter uma única matriz, o OpenGL define uma pilha de matrizes de transformação, cuja posição topo pode ser modificada pelas funções glPushMatrix() e glPopMatrix(). Esse recurso é extremamente útil quando se tem em uma mesma cena um objeto que sofre movimentação relativa a outro

objeto. Como exemplo, pode-se ter um carro que está em cima de um navio. Quando o navio se move, o veículo sofre a mesma transformação, porém nada impende que o veículo ande sobre o navio já em movimento.

Outros usos das transformações de Escala e Translação

Muitas vezes é necessário fazer a transformação de dados para um correto processamento ou exibição em um dispositivo.

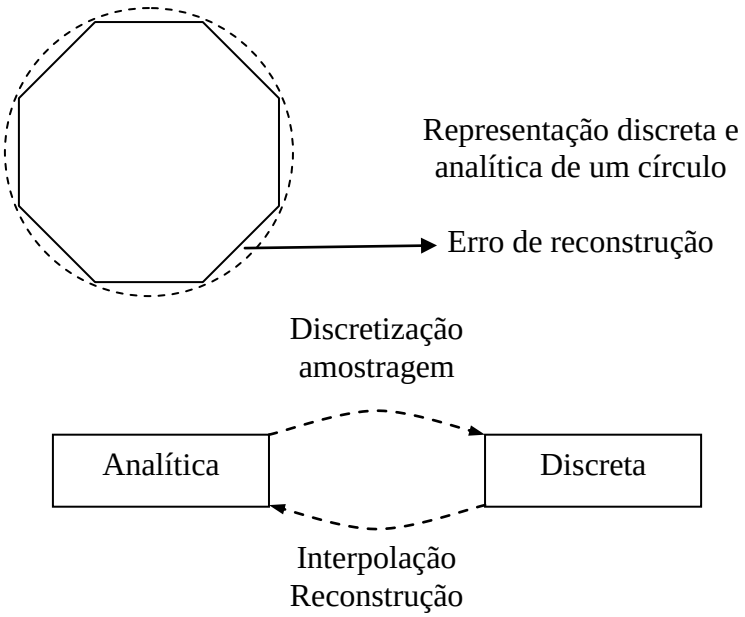
Exemplo 1:
Considere por exemplo que seja necessário armazenar vetores de dados sinalizados em vetores não sinalizados. Considerando um tipo char, em um intervalo sinalizado temos valores entre [-128, 127]. Aplicando-se uma translação nos valores de 128, temos os valores no intervalo [0, 255]. Adicionalmente pode-se aplicar uma escala nos valores, por exemplo, de 0.5, para transformar os valores no intervalo [0, 128]. O processo inverso pode ser usado para levar os valores ao intervalo original.

Exemplo 2:
Suponha que se deseje desenhar na tela um círculo paramétrico (em coordenadas polares). As funções seno e cosseno geram valores entre [-1. 1]. Na tela, tem-se, por exemplo, valores entre [0, 1024]. Para se desenhar um círculo em tela cheia, deve-se fazer as seguintes transformações:
- translação em 1 para levar o intervalo entre [0, 2]
- escala em 512 para levar o intervalo entre [0, 1024]

2 Primitivas Gráficas 2D

Formas de Representação de primitivas:

- Analítica (função):
  - Gasta menos espaço
  - Maior precisão (resolução)
  - Facilidade para calculo de pontos intermediários
  - Representação de Figuras vetoriais.
  - Ex: MS Word.
- Discreta (pontos/pixels - **Amostragem**)
  - Usado para fazer a representação gráfica em dispositivos, como monitores ou impressoras.
  - Representações poligonais.
  - Ex: Modelos de personagens em jogos 3D, imagens BMP, arquivo de impressão, figuras na tela do computador



Representação analítica:

1. **Paramétrica**: cada coordenada de um ponto é representado como uma função de um único parâmetro.
2. **Não Paramétrica**
   a. Explícita → $y = mx + b$
      para cada x, obtém-se apenas 1 valor de *y*. Ex: retas, senoides, etc.
   b. Implícita → $x^2 + y^2 = r^2$
      para cada x, obtém-se 2 valores para *y*. Pode ser usado para representar círculos.

Linha

- Usos
- Geralmente disponibilizado em APIs (Java, OpenGL, Directx, etc)
- Representação paramétrica ou por função

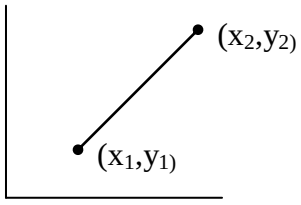
Representação por função Explícita

$y = mx + b$
m = inclinação da reta
b = interseção com o eixo *y*.

$$m = \frac{\Delta y}{\Lambda x} = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_1 - mx_1$$

$$\Delta y = m\Lambda x, \quad \Lambda x = \frac{\Lambda y}{m}$$



Algoritmo DDA (*Digital Differiential Analyser*)

$$y_{k+1} = y_k + m, \quad 0 < m < 1$$

$$x_{k+1} = x_k + \frac{1}{m}, \quad m > 1$$

Algoritmo DDA

```
dx = x2 - x1
dy = y2 - y1
if (|dx| > |dy|)
    steps = |dx|
else
   steps = |dy|
incx = dx/steps
incy = dy/steps

pixel(x1, y1)
for(k=0; k< steps; k++)
{
   x += incx;
   y += incy
   pixel(x, y)
}
```

Representação Paramétrica

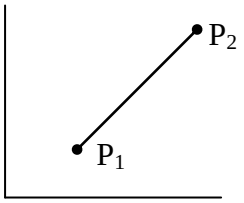
x = x(t)
y = y(t)
P(t) = (x(t), y(t)) *Vetor direção*

$P(t) = P_1 + t(P_2 - P_1), \quad 0 \le t \le 1$ (Uma multiplicação)
$P(t) = P_1 + tP_2 - tP_1$
$P(t) = (1-t)P_1 + tP_2$ (Duas multiplicações)

ou

$P(t) = P_0 + tV, \quad t \ge 0$ (representação de um raio)

se *t=0, P(t) = P₁*
se *t=1, P(t) = P₂*
se *t=0.5, P(t) = 0.5P₁ + 0.5P₂*

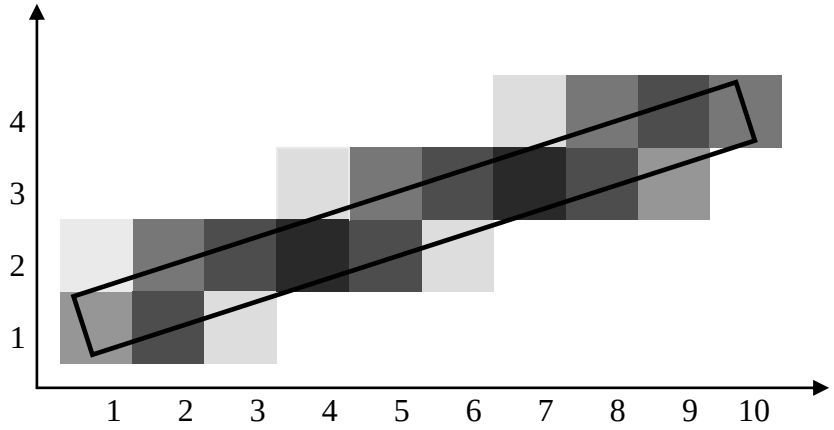


Independente da representação, deve-se observar que quando a linha é representada graficamente por um conjunto de pontos amostrados.

- Dependendo da resolução do dispositivo, pode ser visível a formação de bordas dentadas (*Alias*).
- Uma solução para evitar este problema, espacialmente em monitores onde a resolução é menor que 100 DPI, deve-se utilizar algum algoritmo de *anti-aliasing*.

Na seguinte figura é ilustrado como um algoritmo de **anti-aliasing** pode ser usado para gerar linhas que se pareçam mais suaves.

- Faz uso de pixels com intensidades diferentes, dependendo de sua área de projeção sobre a linha.
- Pixels que são cobertos pela linha tem cor mais escura e pixels que são parcialmente cobertos pela linha tem cor mais clara.

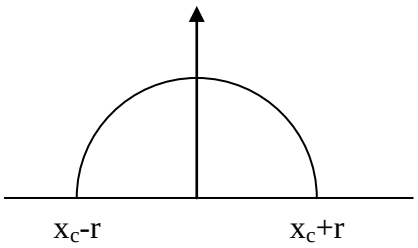


Círculo

Função implícita do círculo

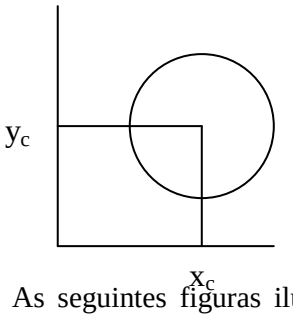
$x^2 + y^2 = r^2$

$$y^2 = r^2 - x^2$$
$$y = \pm\sqrt{r^2 - x^2}$$



Não centrado na origem

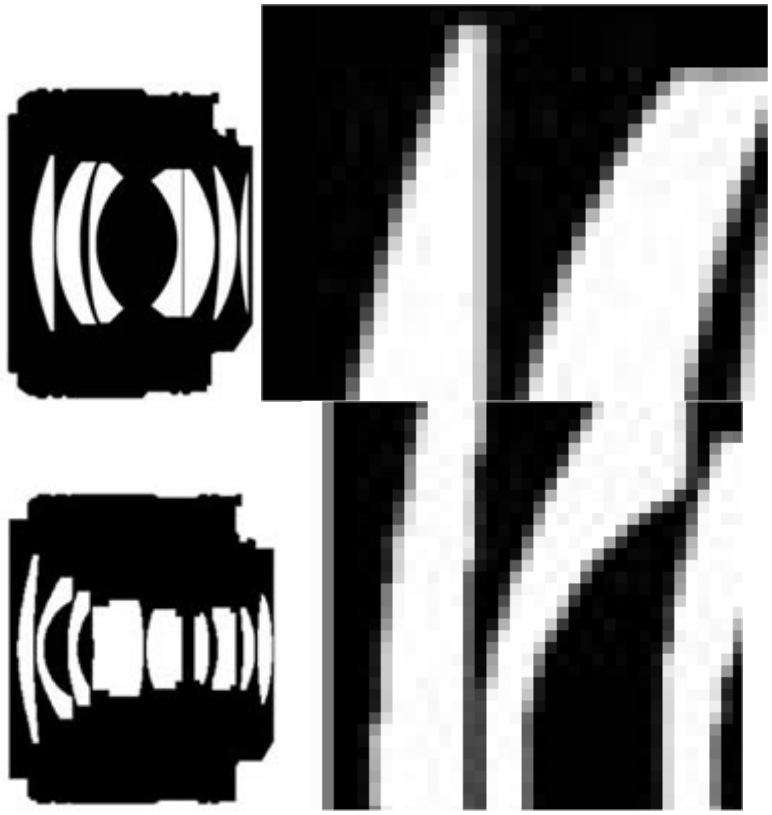
$$(x - x_c)^2 + (y - y_c)^2 = r^2, \quad (x_c - r) \le x \le (x_c + r)$$
$$(y - y_c)^2 = r^2 - (x - x_c)^2$$
$$(y - y_c) = \sqrt{r^2 - (x - x_c)^2}$$
$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2}$$



Equação paramétrica do círculo

$x = x_c + r\cos\theta$ (variação **constante** entre pontos na curva)
$y = y_c + r\sin\theta$

***Anti-aliasing* para círculos:** O processo é o mesmo utilizado para retas. As seguintes figuras ilustram curvas geradas com dois tipos de algoritmos de *anti-aliasing* (imagens de lentes de câmeras - http://nikonimaging.com/global/products/lens/af/dx/index.htm). As figuras da direita são ampliações das figuras da esquerda. Pode-se claramente notar a diferença de qualidade das imagens. Deve-se observar que se estas imagens fossem geradas sem algoritmos de *anti-aliasing*, o resultado seria bem pior.



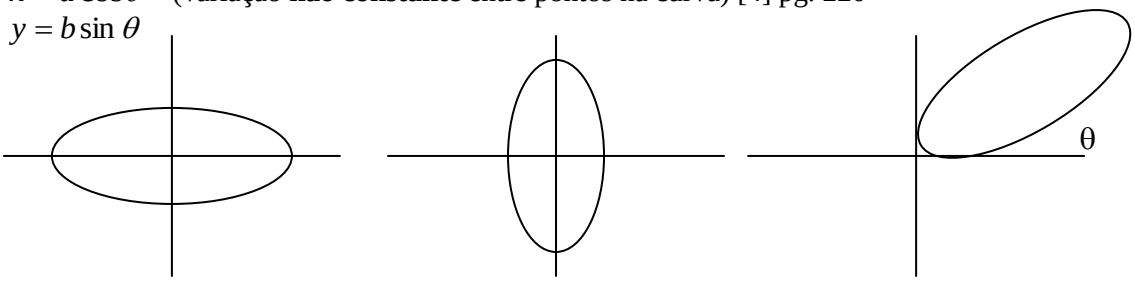
Bom algoritmo de *anti-aliasing*

Algoritmo ruim de *anti-aliasing*

Assim como primitivas, algoritmos de anti-aliasing também são muito utilizados para geração de caracteres. A seguinte figura ilustra a geração de caracteres gerados pelo editor Word. Pode-se observar a adição de pixels coloridos nas bordas dos caracteres.

Elipse

$x = a\cos\theta$ (variação **não constante** entre pontos na curva) [4] pg. 220
$y = b\sin\theta$



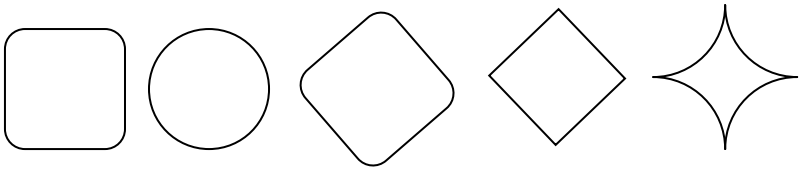
Superquádricas

Superelípse

$$\left(\frac{x}{r_x}\right)^{\frac{2}{s}} + \left(\frac{y}{r_y}\right)^{\frac{2}{s}} = 1$$

paramétrica

$x = r_x \cos^s \theta$
$y = r_y \sin^s \theta$



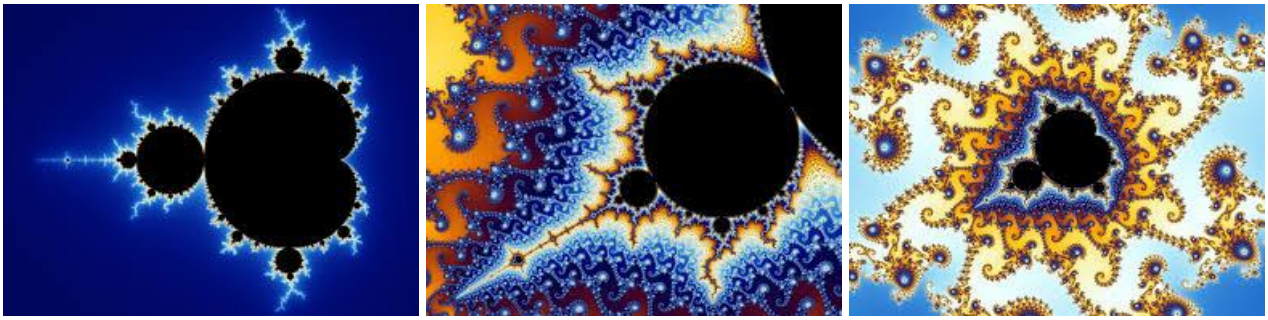
s=0.5 s=1 s=1.5 s=2 s=3

O seguinte algoritmo ilustra a geração paramétrica de superquádricas

```
for(float ang = 0; ang < 2*PI; ang+=0.1)
{
    float x, y;
    x = rx * pow(fabs(cos(ang)), s) * (cos(ang) > 0 ? 1 : -1 );
    y = ry * pow(fabs(sin(ang)), s) * (sin(ang) > 0 ? 1 : -1 );
}
```

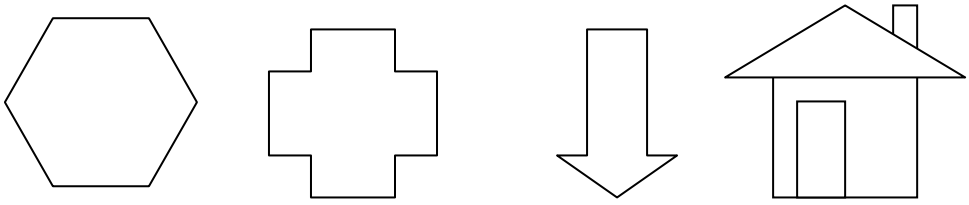
Fractal de Mandelbrot

Definido recursivamente como
$z_0 = 0$
$z_{n+1} = z_n^2 + c$



Poligonal

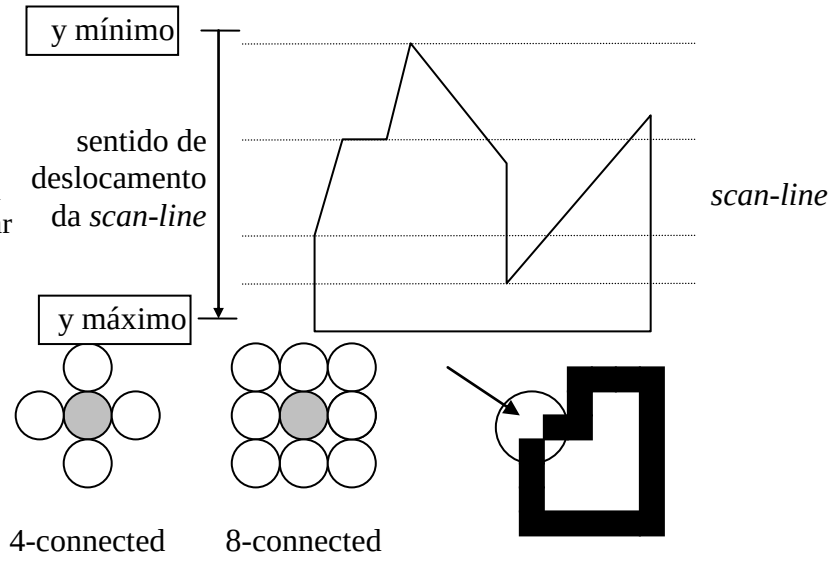
Vetor ordenado de coordenadas que definem a forma do objeto. Pode ser usado para representar qualquer figura. Apresenta limitações para desenho de curvas.



Preenchimento de Polígonos

Scan-line

Faz o preenchimento da cena de linha em linha
Faz uso da inclinação de cada reta para otimizar o processo.



Flood/Boundary fill

Contorno → boundary fill
Cor de fundo → flood fill

```
void boundaryFill4(int x, int y, int fillColor, int boundaryColor)
{
   if(getPixel(x, y)!=boundaryColor && getPixel(x, y)!=fillColor)
   {
      setPixel(x, y, fillColor)
      boundaryFill4(x+1, y, fillColor, boundaryColor)
      boundaryFill4(x-1, y, fillColor, boundaryColor)
      boundaryFill4(x, y+1, fillColor, boundaryColor)
      boundaryFill4(x, y-1, fillColor, boundaryColor)
   }
}

void floodFill4(int x, int y, int fillColor, int oldColor)
{
   if(getPixel(x, y) == oldColor)
   {
      setPixel(x, y, fillColor)
      floodFill4(x+1, y, fillColor, oldColor)
      floodFill4(x-1, y, fillColor, oldColor)
      floodFill4(x, y+1, fillColor, oldColor)
      floodFill4(x, y-1, fillColor, oldColor)
   }
}
```

3 Visualização Bidimensional

Window→ área selecionada do sistema de coordenadas do mundo para ser exibida no display (área que será vista). Deve-se observar que nem toda área de um ambiente pode ser exibida em uma única *window*. Esta área é determinada pela configuração e posicionamento da câmera sintética.

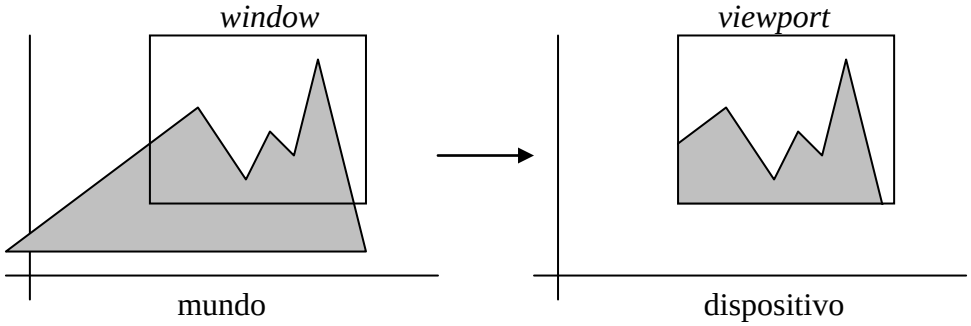
Viewport→ área do display (monitor) na qual a *window* é mapeada. Corresponde a posição na tela onde mostrar. Ex: janela gráfica de uma aplicação gráfica.

Viewing transformation (ou *windowing transformation*)→ mapeamento de uma parte da cena em coordenadas do mundo para coordenadas do dispositivo.

- Deve-se observar que as coordenadas do mundo podem estar no intervalo entre [1, 1] e que as coordenadas da *viewport* sempre estão em resolução de tela.
- A principal transformação é a escala, que pode não ser uniforme.

Passos:

- Definir a área da *window*
- Recortar o que está fora (*clipping*)
- Ajustar posição e escala dentro da *viewport* (mapeamento)

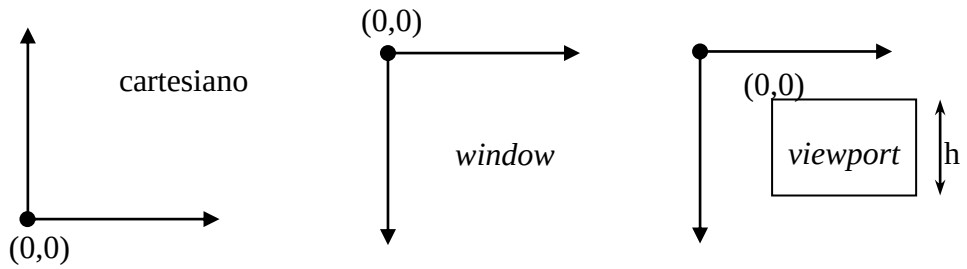


Algoritmos de Clipping

- Linhas (Cohen-Sutherland)
- Polígonos (Weiler-Atherton)
- curvas
- strings

Mapeamento na tela

- Sistema cartesiano define origem no canto inferior esquerdo
- Sistema de tela define a origem no canto superior esquerdo
    - Ex: OpenGL/Glut, Java, etc.



$$x_{tela} = x_{cartesiano}$$

$$y_{tela} = h - y_{cartesiano}$$

Exercícios:

1. Implemente um programa para aplicar matrizes de transformação sobre um quadrado para:
    a. Fazer a rotação
    b. Fazer a escala
    c. Fazer translação
2. Implemente um programa para desenho de linhas com o uso do mouse.
3. Implemente programa para desenho de círculos, elipses e superquádricas.
4. Implemente um editor de figuras com recursos para inserção, movimentação, remoção, rotação, agrupamento, salvamento em disco, etc.
5. Implemente um programa para preenchimento de polígonos com algoritmo flood fill. Faça a mesma solução com algoritmo de scan-line.

Referências:

[1] Lengyel, E.. **Mathematics for 3D Game Programming and Computer Graphics**. Charles River Media, 2002.

[2] Hearn, D., Baker, M. P. **Computer Graphics, C Version** (2nd Edition), Prentice Hall, New Jersey, 1997

[3] Foley, J.D., van Dam, A., Feiner, S.K. and Hughes, J.F. **Computer Graphics: Principles and Practice in C** (2nd Edition), Addison-Wesley Pub. Co., Reading, MA, 1995.

[4] Watt, A. **3D Computer Graphics,** Addison Wesley; 3 edition,1999.

[5] Rogers, D., Adams, J. Mathematical Elements for Computer Graphics, 2nd Edition. Mc Graw Hill, 1990.