LAB. 3

Parte 1

TRANSFORMAÇÕES GEOMÉTRICAS

TRANSFORMAÇÕES GEOMÉTRICAS

As transformações geométricas em 2D possuem generalização de forma direta para 3D. Essas transformações geométricas também podem ser chamadas de transformações afins.

1. Após finalizar este laboratório, responda no seu documento compartilhado as seguintes questões:

1. Explique o que são transformações e por que as usamos em computação gráfica.
2. Liste as três principais transformações que usamos na computação gráfica e descreva o que cada um faz.
3. Entenda e explique como girar um ponto em torno de um ponto arbitrário.
4. Entenda e explique o que são coordenadas homogêneas e por que as usamos em Computação gráfica.
5. Qual a importância da ordem das operações em uma multiplicação de matrizes.
6. O que é uma CTM (Matriz de Transformação Combinada) e explique em que ordem as transformações devem estar para alcançar o desejado CTM.

2. Primeiros passos

As transformações geométricas são usadas para cumprir dois requisitos principais em computação gráfica:

1. **Modelar** e construir cenas.
2. **Navegar** no espaço bidimensional e tridimensional.

Por exemplo, quando um prédio na mesma rua tem n janelas idênticas, procedemos da seguinte forma:

1. Construímos uma única janela por meio de primitivas gráficas;
2. Replicamos n vezes a janela.
3. Colocamos cada janela em um local desejável usando translações e rotações.

Isso mostra que transformações como translações e rotações podem ser usadas como operações de modelagem de cena.

3. Transformações Euclidianas

Existem duas transformações euclidianas:

1. Translação
2. Rotação

3.1. Translação

A translação pode ser considerada como um movimento. Na translação, um ponto é movido a uma distância em uma direção.

Por exemplo, quando o ponto A(x, y) é transladado *dx* unidades na direção *x* e *dy* unidades na direção *y*, torna-se:

$$A'(x + dx, y + dy)$$

ou equivalente a,

$$\begin{cases} x' = x + dx \\ y' = y + dy \end{cases}$$

Representando pontos como matrizes de coluna, obtemos:

$$A = \begin{bmatrix} x \\ y \end{bmatrix}, \qquad A' = \begin{bmatrix} x' \\ y' \end{bmatrix}, \qquad \text{and} \qquad T = \begin{bmatrix} dx \\ dy \end{bmatrix},$$

para que a translação possa ser expressa da seguinte forma:

$$A' = A + T$$

Em geral, transladar um objeto significa transladar seus vértices (ou seja, cantos ou pontos finais) de forma que linhas ou triângulos possam ser desenhados usando os vértices transformados.

3.2. Rotação sobre a origem

Usando coordenadas polares (r, θ), um determinado ponto no plano é dado pela seguinte equações:

$$\begin{cases} x = r \cos \phi \\ y = r \sin \phi \end{cases}$$

Por padrão, girar um objeto pelo ângulo Ø significa girá-lo em torno da origem por Ø. Girando o ponto anterior pelo ângulo Ø em torno da origem, obterá o seguinte ponto transformado:

$$\begin{cases} x' = r\cos(\phi + \theta) \\ y = r\sin(\phi + \theta) \end{cases}$$

Ou

$$\begin{cases} x' = r\cos\phi\cos\theta - r\sin\phi\sin\theta) \\ y' = r\cos\phi\sin\theta + r\sin\phi\cos\theta) \end{cases}$$

Isto é

$$\begin{cases} x' = x\cos\theta - y\sin\theta \\ y' = x\sin\theta + y\cos\theta \end{cases}$$

Na notação de matriz, obtemos então

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

Ou

$$A' = R.A$$

onde R é a matriz de rotação 2x2.

3.3. Coordenadas homogêneas

Vimos as seguintes transformações de matriz:

Translação: $A' = A + T$

Rotação:

$$A' = R.A$$

A translação é obtida através da adição de matriz, enquanto a rotação é alcançada pelo produto entre matrizes. Isso significa que podemos combinar qualquer número de matrizes de translação por meio da adição e qualquer número de matrizes de rotação por meio da multiplicação. No entanto, não podemos combinar matrizes de translação e rotação em uma única matriz por meio da operação produto. Seria muito útil se pudéssemos fazer isso, porque isso permitiria a composição de transformações geométricas por meio de uma única operação de matriz, o produto entre matrizes..

As coordenadas homogêneas são apenas uma forma de superar esse problema. Com coordenadas homogêneas, uma série de transformações geométricas podem ser aplicadas em uma sequência usando o produto de matrizes. O resultado é geralmente chamado de matriz de transformação combinada ou CTM.

Portanto, as translações e rotações expressas em coordenadas homogêneas são dadas por:

Translação: $A' = T.A$

Rotação:

$$A' = R.A$$

Em coordenadas homogêneas, um ponto P (x, y) é representado pelo ponto homogêneo P (x, y, w) onde:

$$X = \frac{x}{W}$$

e

$$Y = \frac{y}{W'}$$

onde W geralmente é igual a 1 em computação gráfica para simplificar.

Usando coordenadas homogêneas, as matrizes de transformação são expressas como Matrizes 3x3 da seguinte forma:

Translação:

$$T(dx, dy) = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$$

Rotação:

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

3.4. Rotação sobre um ponto arbitrário

A matriz de rotação (acima) funciona bem se pretendemos girar um ponto em torno da origem. Mas, que tal girar o ponto (x, y) em torno do ponto arbitrário (x_a, y_a)?

A resposta está no seguinte procedimento de três etapas:
- Transladar (x_a, y_a) para a origem, ou seja, transladar por T ($-x_a$, $-y_a$).
- Executar a rotação $R(\emptyset)$.
- Transladar para que o ponto na origem retorne ao local original, ou seja, traduzir por T (x_a, y_a).

Portanto, para girar um objeto composto de 5 vértices, cada transformação geométrica seria preciso ser feito 5 vezes. No geral, temos 3 transformações x 5 vértices = 15 cálculos o que é computacionalmente caro.

O custo computacional pode ser reduzido usando o CTM (Transformação Combinada de Matrizes), ou seja, combinando as transformações em um único CTM:

3 Transformações x Matriz de Identidade = 3 cálculos

1 Transformação (CTM) x 5 vértices = 5 cálculos

de modo que o número total de cálculos seja igual a 8.

3.5.Ordem e composição das transformações

A ordem das transformações geométricas do CTM é relevante porque o produto da matriz não é comutativo.
Na verdade, o produto de matrizes é *associativo*:

Ao multiplicar matrizes, a ordem que realizamos as multiplicações não é relevante, que é

$$A.B.C = (A.B).C = A.(B.C)$$

Porém, a multiplicação da matriz não é *comutativa*:

Ao multiplicar matrizes, a realização das multiplicações é relevante, isto é

$$A.B \neq B.A$$

A questão é, então, como calculamos a ordem de nossas matrizes ao criar o CTM?

Voltando às etapas para girar um ponto em torno do ponto T $(-x_a, -y_a)$, vamos reescrever o procedimento correspondente:

- Transladar (x_a, y_a) para a origem, ou seja, transladar por T $(-x_a, -y_a)$.
- Executar a rotação $R(\emptyset)$.
- Transladar para que o ponto na origem retorne ao local original, ou seja, traduzir por T (x_a, y_a).

Assim, a ordem do CTM é:

$$CTM = T\ (x_a, y_a).\ R(\emptyset).\ T\ (-x_a, -y_a)$$

Quando multiplicamos o CTM pelo ponto P, temos

$$CTM.P = T\ (x_a, y_a).\ R(\emptyset).\ T\ (-x_a, -y_a)\ .P$$

Um fato importante a se ter em mente é que a transformação mais próxima do ponto P na expressão é a primeira transformação a ser aplicada a P.

4. Transformações afins

As transformações euclidianas preservam a distância entre os pontos e, por isso, são então chamadas de transformações rígidas.

As transformações afins generalizam as transformações euclidianas no sentido de que não preserve a distância, mas sim o paralelismo. Isso significa que duas linhas paralelas permanecem paralelas após a aplicação de uma transformação afim. Por causa dessa invariante principal, outras propriedades são preservadas. Por exemplo, uma transformação afim também preserva a colinearidade (ou seja, todos os pontos de uma linha permanecem em uma linha após a transformação) e razões de distâncias ou proporções (por exemplo, o ponto médio de um segmento de linha permanece o ponto médio após a transformação).

Uma transformação afim também é chamada de afinidade. Exemplos de transformações afins são contração, expansão, dilatação, reflexão, rotação, cisalhamento, transformações de similaridade e translação, assim como suas combinações. Em geral, uma transformação afim é o resultado de uma composição de rotações, translações, dilatações e cisalhamento.

As rotações e translações são transformações euclidianas. Vamos então ver outra transformação afim básica.

Dimensionamento ou escala:

No dimensionamento, alteramos o tamanho de um objeto. O dimensionamento torna um objeto maior ou menor em direção a x e/ou y.

Escalar um ponto (x, y) por um fator s_x ao longo do eixo x e x_y ao longo do eixo y requer que seja multiplicada cada coordenada pelo fator de escala correspondente:

$$\begin{cases} x' = s_x . x \\ y' = s_y . y \end{cases}$$

Ou, usando notação de matrizes,

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} . \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

5. Transformações 3D em OpenGL

No OpenGL moderno, as transformações geométricas são definidas usando GLM (OpenGL Mathematics) e GLSL (OpenGL Shading Language) da seguinte forma:

$$\underset{\text{myMatrix}}{\begin{bmatrix} a & b & c & d \\ e & f & h & i \\ j & k & l & m \\ n & o & p & q \end{bmatrix}} \times \underset{\text{myVector}}{\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}} = \underset{\text{transformedVector}}{\begin{bmatrix} ax + by + cz + dw \\ ex + fy + hz + iw \\ jx + ky + lz + mw \\ nx + oy + pz + qw \end{bmatrix}}$$

In C++, with GLM:

```
glm::mat4 myMatrix;
glm::vec4 myVector;
// fill myMatrix and myVector somehow
glm::vec4 transformedVector = myMatrix * myVector; // Again, in this order ! this is important.
```

In GLSL :

```
mat4 myMatrix;
vec4 myVector;
// fill myMatrix and myVector somehow
vec4 transformedVector = myMatrix * myVector; // Yeah, it's pretty much the same than GLM
```

Translação

No GLM, a translação é definida como:

```
glm::mat4 glm::translate(
        glm::mat4 const & m,
        glm::vec3 const & translation);
```

que transforma uma matriz com uma matriz de translação *m* 4x4 criada a partir de 3 escalares, conforme o exemplo:

$$\begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 10 \\ 10 \\ 10 \\ 1 \end{bmatrix} = \begin{bmatrix} 20 \\ 10 \\ 10 \\ 1 \end{bmatrix}$$

In C++, with GLM:

```
#include <glm/gtx/transform.hpp> // after <glm/glm.hpp>

glm::mat4 myMatrix = glm::translate(glm::mat4(), glm::vec3(10.0f, 0.0f, 0.0f));
glm::vec4 myVector(10.0f, 10.0f, 10.0f, 0.0f);
glm::vec4 transformedVector = myMatrix * myVector; // guess the result
```

In GLSL :

```
vec4 transformedVector = myMatrix * myVector;
```

Rotação

Em GLM, a rotação é definida como:

```
glm::mat4 glm::rotate(
        glm::mat4 const & m,
        float angle,
        glm::vec3 const & axis);
```

que transforma uma matriz com uma matriz *m* de rotação 4x4 criada a partir de um eixo de 3 escalares e um ângulo expresso em graus.

In C++:

```
// Use #include <glm/gtc/matrix_transform.hpp> and #include <glm/gtx/transform.hpp>
glm:vec3 myRotationAxis( 1.0f, 0.0f, 0.0f );
glm:mat4 rot = glm::rotate( angle_in_degrees, myRotationAxis);
```

Escala

Em GLM, a escala é definida como:

```
glm::mat4 glm::scale(
        glm::mat4 const & m,
        glm::vec3 const & factors);
```

que transforma uma matriz com uma matriz m de escala 4x4 criada a partir de um vetor de 3 componentes.

In C++:

```
// Use #include <glm/gtc/matrix_transform.hpp> and #include <glm/gtx/transform.hpp>
glm:mat4 myScalingMatrix = glm::scale(2.0f, 2.0f, 2.0f );
```

Para maiores detalhes consulte:
http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/

Exercícios:

1. Baixe o programa movingHouse.cpp do moodle e reescreva de forma que todos os edifícios blocos (corpo, telhado, janelas e porta) sejam construídos a partir da origem. Então, use translações para colocar esses blocos de construção nos locais desejados. Cada bloco de construção é construído em uma função separada. No caso da janela, precisamos chamá-la duas vezes porque estamos assumindo que a casa tem duas janelas.
2. Adicione iterações do teclado da seguinte forma: remover o corpo da casa pressionando a tecla 'b', o teto pressionando a tecla 'r', as janelas pressionando a tecla 'w' e a porta pressionando a tecla 'd'.
3. Vamos agora replicar duas vezes a casa. A primeira cópia da casa original deve ser reduzida para ¾ e colocada lado a lado da casa original à esquerda dela. A cópia da segunda casa deve ser dimensionada para 5/4 e colocada lado a lado da casa original à direita dela.
4. Vamos agora adicionar o sol brilhante à cena. O sol pode ser gerado usando o programa referente ao Exercício 5 do Lab 02. O usuário pode alterar a posição do sol clicando na tecla 's'. A trajetória do sol é um arco de círculo.
5. Com base no código original de movingHouse.cpp, faça os elementos da casa (ou seja, telhado, janelas e porta) se afastarem do centro do corpo (pode usar as setas do teclado para isso).

6. Com base no código original de movingHouse.cpp, faça os elementos da casa (ou seja, telhado, janelas e porta) girarem em torno do centro do corpo (pode usar as setas do teclado para isso).
7. Com base no código original de movingHouse.cpp, faça os elementos da casa (ou seja, telhado, janelas e porta) girarem e se afastar do centro do corpo simultaneamente (pode usar as setas do teclado para isso).
8. Insira as teclas ESC para sair e ESPACE para wireframe e fill.

Parte 2

ANIMAÇÃO BÁSICA E COLISÕES

Nesta segunda parte deste laboratório, pretendemos aprender os conceitos básicos de animação e detecção de colisão quando os objetos se movem em uma cena 2D.

1. Objetivos de aprendizagem

Ao final deste laboratório, você será capaz de:

1. Usar transformações geométricas com parâmetros variáveis para animar objetos dentro de uma cena 2D.
2. Detectar colisões de objetos 2D em movimento contra as bordas da cena.
3. Detectar colisões entre objetos em uma cena 2D.

2. Exemplo

Baixe o programa squareColision.cpp do moodle, que exibe um quadrado saltando dentro de um subdomínio com uma colisão básica e a detecção das bordas de tal subdomínio.

3. Exercícios de programação

1. Escreva um programa que mova um quadrado 20,0x20,0 dentro de um domínio 40x30 em R^2. A janela de visualização correspondente tem 800x600 pixels. O movimento do quadrado é realizado em passos de 0,1 na direção x e na direção y. A posição inicial do quadrado está na origem (0.0,0.0).
2. Altere o programa squareColision.cpp para animar dois quadrados que se cruzam perpendicularmente dentro da caixa.
3. Altere o programa squareColision.cpp para não apenas mover o quadrado, mas também girá-lo 2,5 graus a cada passo.
4. Escreva um programa que mova um círculo dentro de uma caixa quadrada. O círculo volta quando atinge qualquer lado da caixa.
5. Escreva um programa para rolar uma roda em uma linha horizontal.
6. Escreva um jogo para Atari com uma bola, uma raquete e uma parede de tijolos.
7. Insira as teclas ESC para sair e ESPACE para wireframe e fill.