Transformações Geométricas e Animação

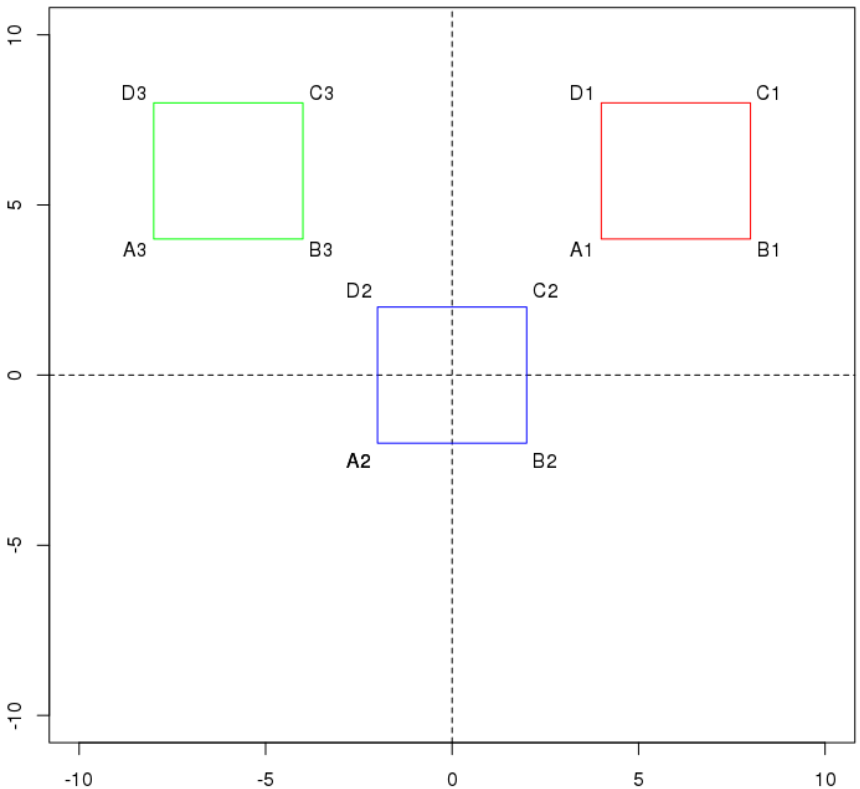
SCC0250/0650 - Computação Gráfica

Prof. Rosane Minghim
https://edisciplinas.usp.br/course/view.php?id=61213
https://edisciplinas.usp.br/course/view.php?id=61210
P.A.E. Diego Cintra e Fábio Felix
diegocintra@usp.br, *f_diasfabio@usp.br*

Instituto de Ciências Matemáticas e de Computação (ICMC)
Universidade de São Paulo (USP)

19 de março de 2018

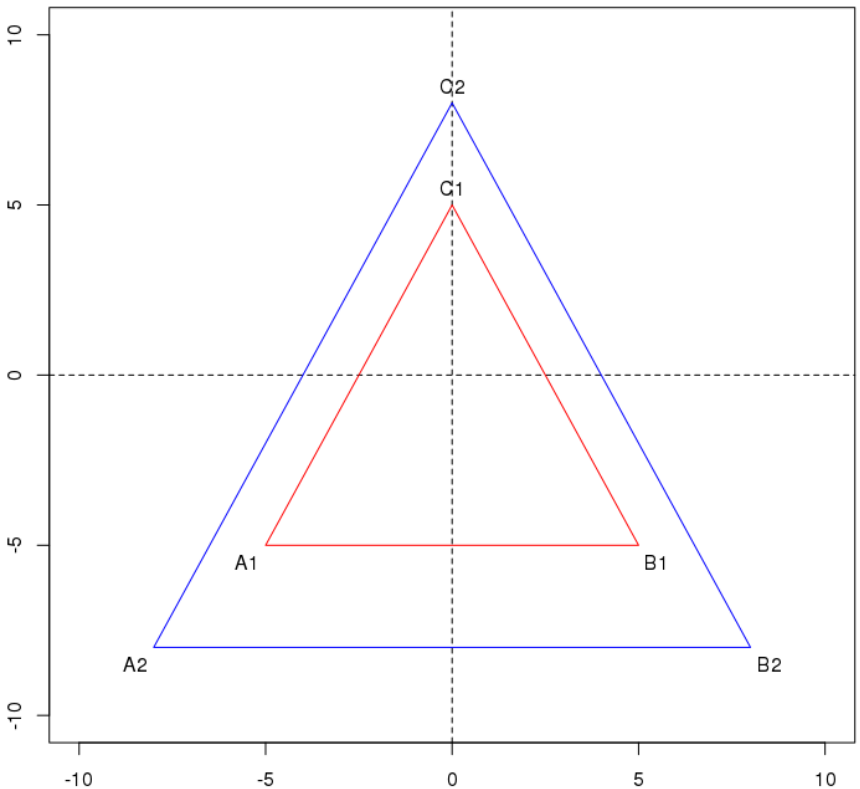
Transformações Geométricas - Translação




```
1  struct Point
2  {
3    GLdouble x, y;
4  };
5  vector<Point> v
6  ...
7  for(int i = 0; i < v.size(); i↩
       ++)
8  {
9    v[i].x += tx;
10   v[i].y += ty;
11 }
```

```
1  void glTranslated(GLdouble x, ↩
       GLdouble y, GLdouble z);
2  void glTranslatef(GLfloat x, ↩
       GLfloat y, GLfloat z);
```

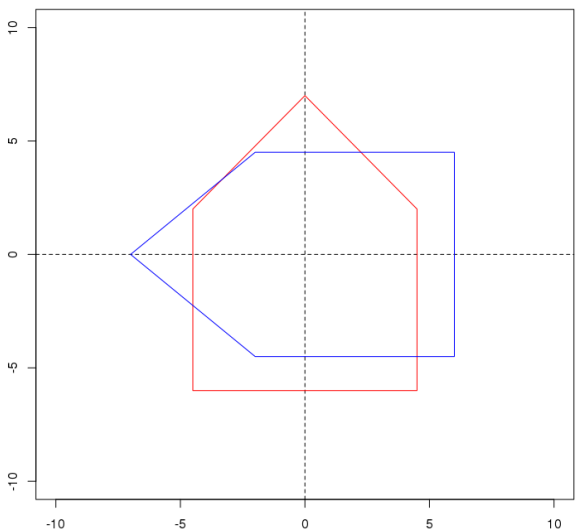
Transformações Geométricas - Escala




```
1  struct Point
2  {
3    GLdouble x, y;
4  };
5  vector<Point> v
6  ...
7  for(int i = 0; i < v.size(); i↩
       ++)
8  {
9    v[i].x *= sx;
10   v[i].y *= sy;
11 }
```

```
1  void glScaled(GLdouble x, ↩
       GLdouble y, GLdouble z);
2  void glScalef(GLfloat x, ↩
       GLfloat y, GLfloat z);
```

Transformações Geométricas - Rotação




```
1  #include<cmath>
2  ...
3  GLdouble x_aux = 0,
4           theta_radi = theta * M_PI / 180.0,
5           cos_th = cos(theta_radi),
6           sin_th = sin(theta_radi);
7
8  for(int i = 0; i < v.size(); i++)
9  {
10   x_aux = v[i].x;
11   v[i].x = x_aux * cos_th - v[i].y * sin_th;
12   v[i].y = x_aux * sin_th + v[i].y * cos_th;
13 }
```

```
1  void glRotated(GLdouble angle, GLdouble x, ↩
       GLdouble y, GLdouble z);
2  void glRotatef(GLfloat angle, GLfloat x, GLfloat↩
        y, GLfloat z);
```

Transformações Geométricas - Usando ponto arbitrário

Escala

```
1 Point fixed_point;
2 ...
3 v[i].x = v[i].x * sx + fixed_point.x * (1 - sx);
4 v[i].y = v[i].y * sy + fixed_point.y * (1 - sy);
```

Rotação

```
1 Point pivot;
2 ...
3 x_aux = v[i].x;
4 v[i].x = pivot.x + (x_aux - pivot.x) * cos_th - (v[i].y - pivot.y) * ↩
      sin_th;
5 v[i].y = pivot.y + (x_aux - pivot.x) * sin_th + (v[i].y - pivot.y) * ↩
      cos_th;
```

Transformações Geométricas - Coordenadas Homogêneas I

Utilizadas para tratar as transformações da mesma maneira, efetuar facilmente composição de transformações e melhorar a eficiência.

$$(x, y) \rightarrow (x_h, y_h, h)$$

$$(x, y, z) \rightarrow (x_h, y_h, z_h, h)$$

$$x = \frac{x_h}{h} \qquad y = \frac{y_h}{h} \qquad z = \frac{z_h}{h} \qquad h = 1$$

Transformações Geométricas - Coordenadas Homogêneas II

Translação 2D

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Escala 2D

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} sx & 0 \\ 0 & sy \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Transformações Geométricas - Coordenadas Homogêneas III

Rotação 2D

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sen(\theta) \\ sen(\theta) & cos(\theta) \end{bmatrix} * \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sen(\theta) & 0 \\ sen(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Transformações Geométricas - Observações

- Todas as transformações da OpenGL consideram a origem como referência. Antes de fazer rotação ou escala é necessário transladar para a origem, efetuar as transformações e transladar de volta para a posição inicial.

```
1 ...
2 glMatrixMode(GL_MODELVIEW);
3 ...
4 glTranslated(tx, ty, 0);
5 glRotated(theta, 0.0, 0.0, 1.0);
6 glTranslated(-tx, -ty, 0);
7 ...
```

Transformações Geométricas - Observações

- Todos os comandos de transformação são compostos em uma matriz de transformação. A OpenGL possui 4 tipos de matrizes *GL_MODELVIEW*, *GL_PROJECTION*, *GL_TEXTURE*, *GL_COLOR*.
- A matriz atual pode ser alterada com *glMatrixMode*.
- Cada novo comando é acumulado, alterando a configuração da matriz atual.
- Ao especificar um novo vértice, a sua posição é calculada aplicando-se a matriz de transformação corrente às suas coordenadas.
- A matriz de transformação é inicializada com a matriz identidade. Na OpenGL pode-se inicializar a matriz com a função *glLoadIdentity()*.

Transformações Geométricas - Observações

- A ordem que as transformações são realizadas influencia no resultado final.

$$T2 * R * T1 \neq T1 * R * T2$$

- Todas as operações da OpenGL são 3D, sendo o 2D um caso particular delas. Por exemplo, para as transformações os valores referentes à coordenada z devem ser considerados como listado abaixo.

```
1 glTranslated(tx, ty, 0);
2 glRotated(theta, 0.0, 0.0, 1.0);
3 glScaled(sx, sy, 1.0);
```

Pilha de Transformações

- Cada modo definido por *glMatrixMode* possui uma pilha de matrizes. A matriz corrente de cada modo é a matriz do topo da sua respectiva pilha.
- A função *glPushMatrix* duplica a matriz do topo da pilha e essa cópia se torna o novo topo da pilha
- A função *glPopMatrix* desempilha a matriz atual do respectivo modo ativo.
- A função *glLoadIdentity* atribuí o valor da matriz identidade à matriz do topo da pilha corrente.

```
1 //Empilha uma copia da matriz atual
2 void glPushMatrix();
3 //Desempilha a matriz atual
4 void glPopMatrix();
5 //Carrega valores da matriz identidade
6 void glLoadIdentity();
```

Pilha de Transformações

```
1  ...
2  glMatrixMode(GL_MODELVIEW);
3  ...
4  glPushMatrix();
5  glTranslated(tx, ty, 0);
6  glRotated(theta, 0.0, 0.0, 1.0);
7  glScale(sx, sy, 1.0);
8  ...
9  //DESENHA ALGUMA COISA
10 ...
11 glPopMatrix();
12 ...
13 //DESENHA OUTRA COISA SEM CONSIDERAR AS
14 //TRANSFORMACOES ANTERIORES
15 ...
```

Pilha de Transformações

- As matrizes também podem ser salvas ou recarregadas, possibilitando que o programador utilize qualquer tipo de combinação de matrizes para compor sua imagem.

```
1 void glGetDoublev(GLenum pname, GLdouble *params);
2 void glGetFloatv(GLenum pname, GLfloat *params);
3 void glLoadMatrixd(const GLdouble *m);
4 void glLoadMatrixf(const GLfloat *m);
```

```
1 GLfloat m[16];
2 ...
3 glGetFloatv(GL_MODELVIEW_MATRIX, m);
4 ...
5 glLoadMatrixf(m);
6 ...
```

- Os arrays seguem o método *column-major order*.

Transformações Hierárquicas

- São transformações diferentes aplicadas em objetos que seguem uma hierarquia.
- Por exemplo, podemos aplicar diferentes transformações sobre partes do corpo humano.
  - Rotacionar o corpo, transladar o braço, entre outras.
- As transformações de mais baixa hierarquia (braços) acumulam as transformações da hierarquia mais alta (corpo).

Animação

- Animação tradicional envolve uma sequência de imagens em alta velocidade.
- Velocidade de exibição (*frame rate*) varia de acordo com a mídia utilizada.
- Cada imagem (quadro, cena) deve possuir uma ligeira diferença em relação às outras, criando a ilusão de movimento.
- As diferenças podem ser na movimentação dos objetos, suar cores, formas etc. Também é possívle modificar a posição do observador quando a imagem for 3D.
- Importante garantir que a aparência da imagem não mude muito rapidamente, pois pode gerar *temporal aliasing*.
  - Exemplo: `http://www.nvidia.com.br/object/txaa-anti-aliasing-technology-br.html`

Animação - OpenGL

- Como a imagem precisa ser modificada continuamente a tela precisa ser atualizada constatemente. Com isso, é necessário evitar que a imagem fique "piscando"quando a tela é redesenhada
- Para evitar esse problema a OpenGL utiliza dois *buffers* para exibição. Enquanto um está sendo preenchido, o outro está sendo exibido.
- O parâmetro GLUT_DOUBLE deve ser utilizado na função *glutInitDisplayMode* para que a OpenGL utilize os dois *buffers*.

Animação - OpenGL

```
1 //Executa o parametro quando nenhum evento esta ocorrendo
2 void glutIdleFunc(void (*func)(void));
3 //Executa a funcao parametro a cada msecs
4 void glutTimerFunc(unsigned int msecs,void (*func)(int value), value);
5 //Alterna os buffers da tela
6 void glutSwapBuffers();
```

Bibliografia

- **Básica:**
  - Hearn, D. Baker, M. P. Computer Graphics with OpenGL, Prentice Hall, 2004. **(livro texto)**
  - Angel, E. Interactive computer graphics: a top-down approach with OpenGL, Addison Wesley, 2000.
  - Foley, J. et. al - Introduction to Computer Graphics, Addison-Wesley, 1993.
  - Kessenich, J., Sellers, G., Shreiner, D. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V - Ninth Edition.

Bibliografia

- **Complementar:**
  - Computer Graphics Comes of Age: An Interview with Andries van Dam. CACM, vol. 27, no. 7. 1982
  - The RenderMan – And the Oscar Goes to... IEEE Spectrum, vol. 38, no. 4, abril de 2001.