COMPUTAÇÃO GRÁFICA – TRANSFORMAÇÕES GEOMÉTRICAS

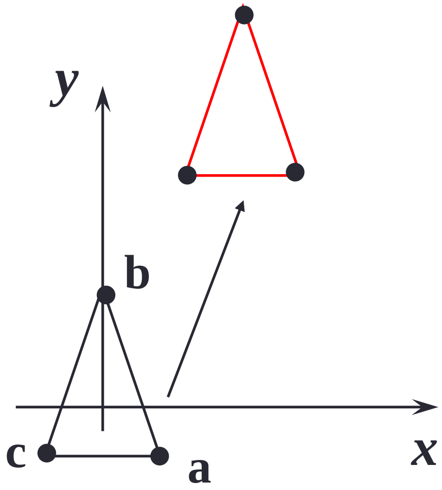
Profa. Mercedes Gonzales Márquez

Tópicos

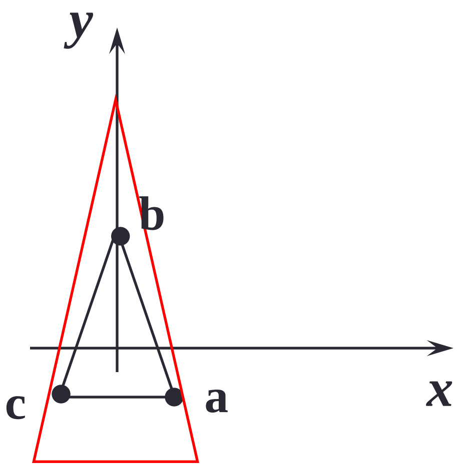
- Transformação Geométrica
- As três transformações geométricas básicas: Translação, Escala e Rotação.

Transformação Geométrica

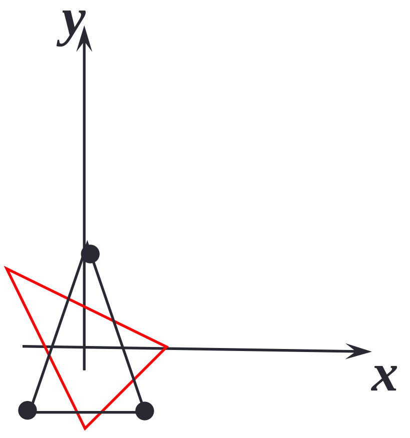
- Transformação que altera algumas características como posição, orientação, forma ou tamanho das figuras geométricas no espaço.
- Apresentamos as três transformações básicas



Translação



Escala



Rotação

Objetos disponíveis

A biblioteca GLUT oferece uma coleção de objetos disponíveis em modo sólido e aramado.

*void **glutWireSphere**(GLdouble radius, GLint slices, GLint stacks);*

- *void **glutSolidSphere**(GLdouble radius, GLint slices, GLint stacks);*
- *void **glutWireCube**(GLdouble size);*
- *void **glutSolidCube**(GLdouble size);*
- *void **glutWireCone**(GLdouble radius, GLdouble height, GLint slices,GLint stacks);*
- *void **glutSolidCone**(idem);*
- *void **glutWireTorus**(GLdouble innerRadius, GLdouble outerRadius,GLint nsides, GLint rings);*
- *void **glutSolidTorus**(GLdouble innerRadius, GLdouble outerRadius,GLint nsides, GLint rings);*

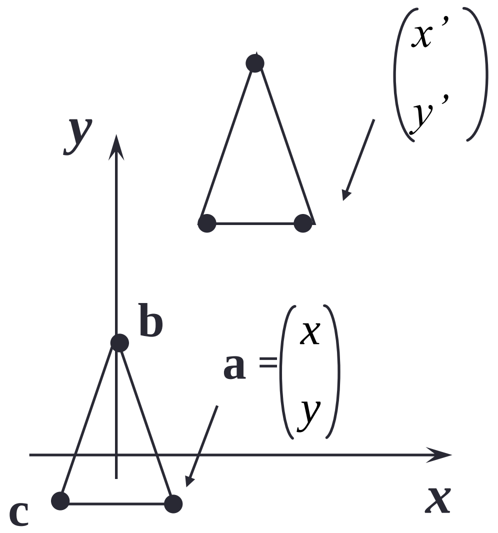
Objetos disponíveis

- *void **glutWireDodecahedron**(GLdouble radius);*
- *void **glutSolidDodecahedron**(GLdouble radius);*
- *void **glutWireOctahedron**(void);*
- *void **glutSolidOctahedron**(void);*
- *void **glutWireTetrahedron**(void);*
- *void **glutSolidTetrahedron**(void);*
- *void **glutWireIcosahedron**(void);*
- *void **glutSolidIcosahedron**(void);*
- *void **glutWireTeapot**(GLdouble size);*
- *void **glutSolidTeapot**(GLdouble size);*

Veja e rode o programa glutObjects.cpp

Transformações lineares: Translação

Transladar significa movimentar o objeto. Transladamos um objeto transladando todos os seus pontos. Para obter a partir de um ponto (x,y) um novo ponto (x’,y’) no plano adicionamos quantidades às suas coordenadas.

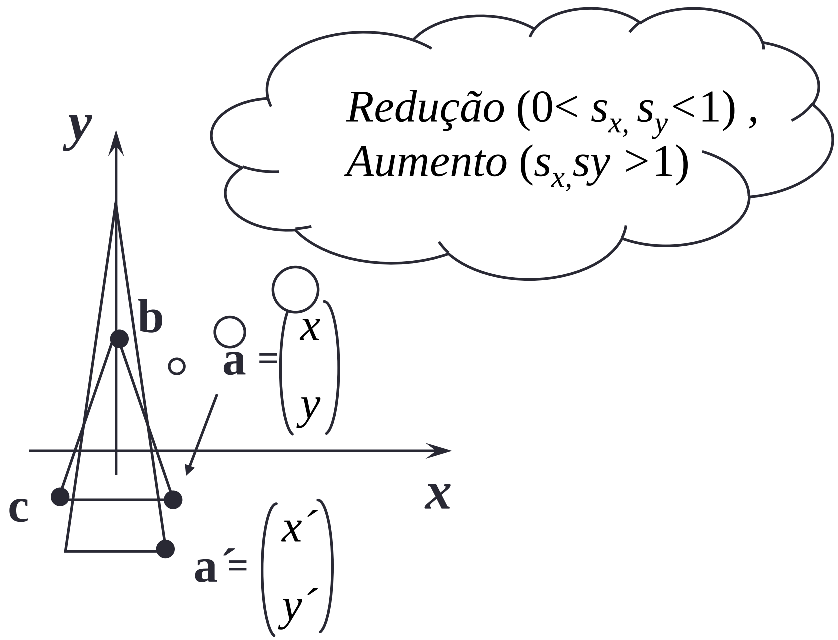


$$x' = x + t_x, \quad y' = y + t_y$$

Veja o programa box.cpp.

Transformações lineares: Escala

Escalar significa mudar as dimensões de escala. Para isso multiplicamos os valores de suas coordenadas por um fator de escala.



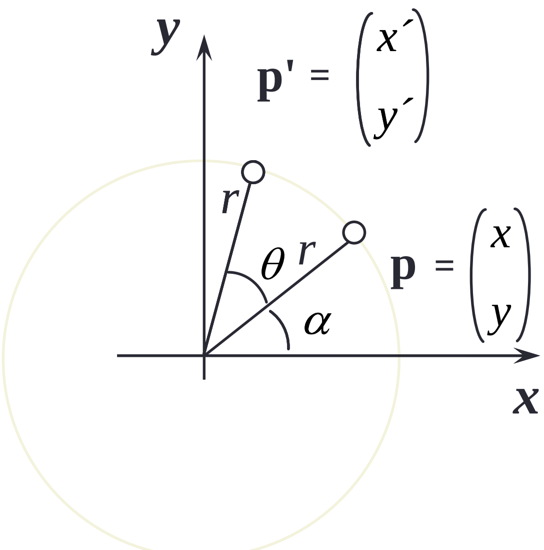
$$x' = s_x x, \quad y' = s_y y$$

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

$$\mathbf{S} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$$

Transformações lineares: Rotação

Rotacionar significa girar. Na Figura abaixo mostra-se a rotação de um ponto p em torno da origem (0,0), passando para a posição p'.



$$\sin(\alpha+\theta) = \sin\alpha \cdot \cos\theta + \cos\alpha \cdot \sin\theta$$

$$\cos(\alpha+\theta) = \cos\alpha \cdot \cos\theta - \sin\alpha \cdot \sin\theta$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} r\cos(\alpha+\theta) \\ r\sin(\alpha+\theta) \end{pmatrix} = \begin{pmatrix} r\cos\alpha \cdot \cos\theta - r\sin\alpha\sin\theta \\ r\sin\alpha \cdot \cos\theta + r\cos\alpha \cdot \sin\theta \end{pmatrix}$$

$$x' = x.\cos\,\theta - y.sen\,\theta$$
$$y' = x.sen\,\theta + y.\cos\,\theta$$

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

Matriz de rotação no plano xy por um ângulo Θ

Resumo - Transformações 2D

Translação

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

Escala

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

Rotação

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

Transformações 3D

Translação

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

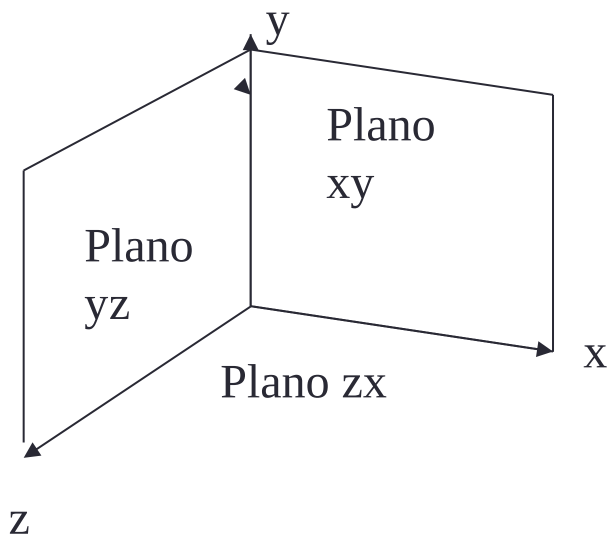
Escala

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Rotação ao redor do eixo z

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

Rotações 3D



$$R_z(\theta):\begin{bmatrix}\cos\theta & -sen\theta & 0\\ sen\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}$$

$$R_y(\theta):\begin{bmatrix}\cos\theta & 0 & sen\theta\\ 0 & 1 & 0\\ -sen\theta & 0 & \cos\theta\end{bmatrix}$$

$$R_x(\theta):\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta & -sen\theta\\ 0 & sen\theta & \cos\theta\end{bmatrix}$$

Coordenadas homogêneas

- Translação não é linear. Como representar em forma de matriz?

  $x' = x + tx \quad y' = y + ty \quad z' = z + tz$

- Solução: uso de coordenadas homogêneas

Coordenadas Homogêneas

- Adiciona uma terceira coordenada w.

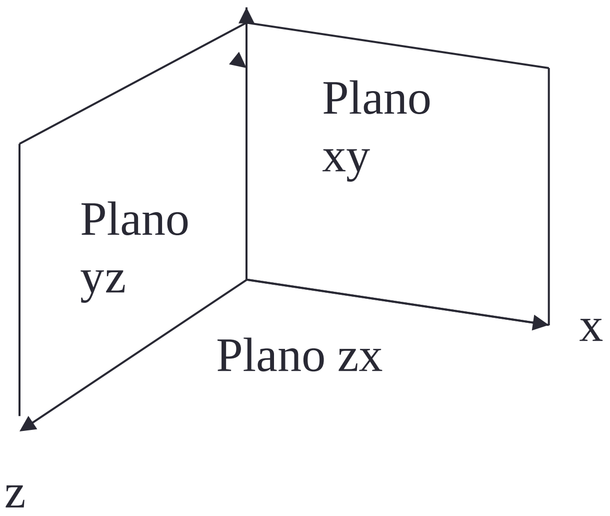
- Um ponto 2D passa a ser um vetor com 3 coordenadas
- Uma transformação do sistema homogêneo para o cartesiano se dá pela seguinte relação: (x’,y’)=(x/w,y/w)
- W=1 a transformação entre os espaços é direta de modo que, (x,y,1) no sistema homogêneo tem os mesmos valores no espaço cartesiano 2D: (x,y).

Transformações 3D

- Translação – glTranslatef(dx, dy, dz)
  - T(dx, dy, dz): $\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$
- Escala – glScalef(Sx, Sy, Sz)
  - S(Sx, Sy, Sz): $\begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Transformações 3D

Rotação :
glRotatef(angle,x,y,z)



$$R_z(\theta):\begin{bmatrix}\cos\theta & -sen\theta & 0 & 0\\ sen\theta & \cos\theta & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$R_y(\theta):\begin{bmatrix}\cos\theta & 0 & sen\theta & 0\\ 0 & 1 & 0 & 0\\ -sen\theta & 0 & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$R_x(\theta):\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & \cos\theta & -sen\theta & 0\\ 0 & sen\theta & \cos\theta & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Transformações em OpenGL

Experimento: Adicione um comando de escala no programa box.cpp. Assim:

```
//Modeling transformations
glTranslatef(0.0,0.0,-15.0); /*Leva o objeto dentro do v.visualização*/
glScalef(2.0,3.0,1.0);
```

Experimento: Um objeto menos simétrico é mais interessante para trabalhar as transformações. Por exemplo o teapot. Troque o cubo pela chaleira, da seguinte forma:

```
//Modeling transformations
glTranslatef(0.0,0.0,-15.0);
glScalef(1.0,1.0,1.0);
glutWireTeapot(5.0);
```

Transformações em OpenGL

Mude sucessivamente os parâmetros da escala substituindo-os pelos seguintes:

1. glScalef (2.0,1.0,1.0)
2. glScalef (1.0,2.0,1.0)
3. glScalef(1.0,1.0,2.0)

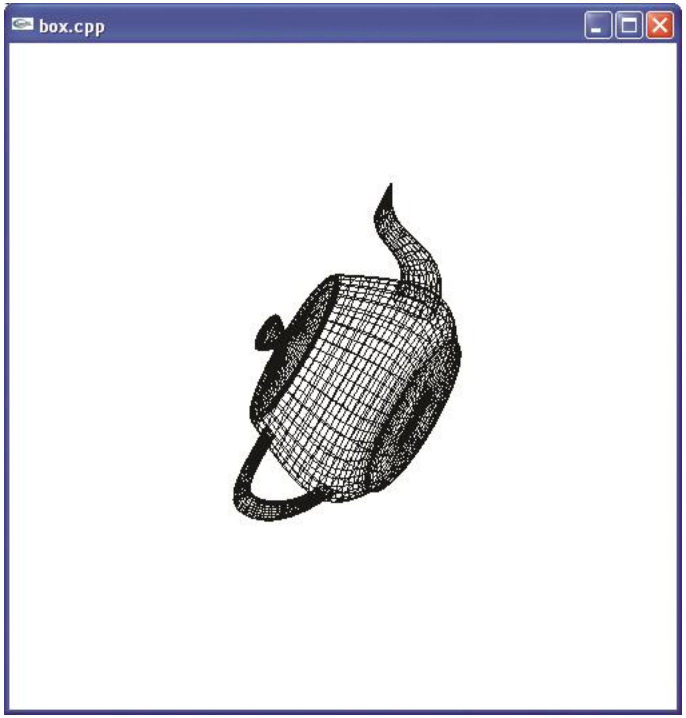
Exercício: A transformação (x,y,z)->(-x,y,z) é uma reflexão (espelhamento) em relação ao plano yz.

4. glScalef(-1.0,1.0,1.0)
5. glScalef(1.0,-1.0,1.0)
6. glScalef(1.0,1.0,-1.0)
7. glScalef(-1.0,-1.0,1.0)

Transformações em OpenGL

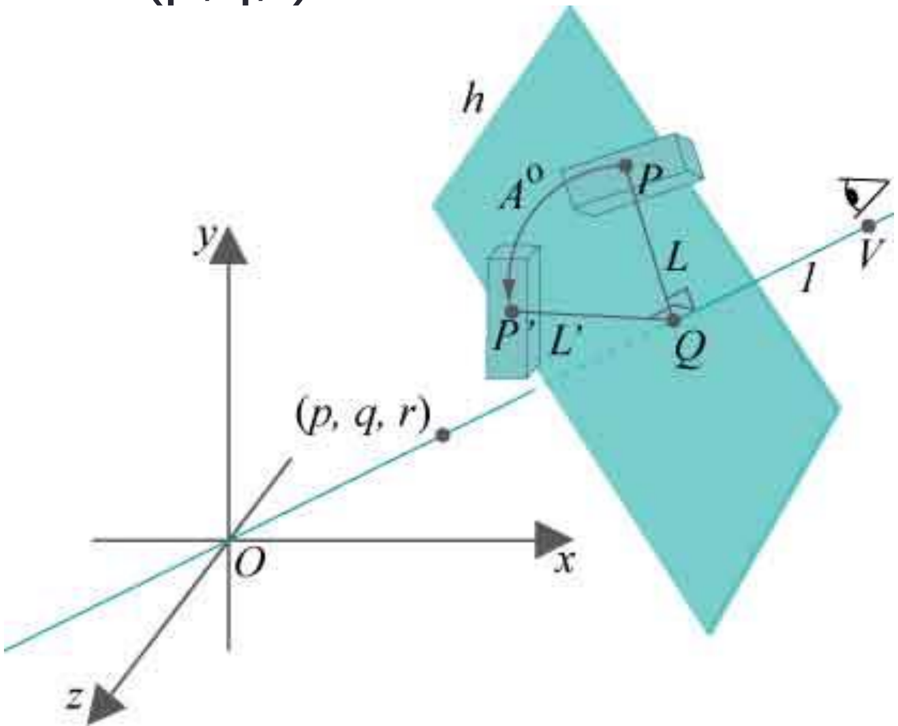
Experimento: Troque o comando de escala pelo seguinte comando de rotação em box.cpp:

```
//Modeling transformations
glTranslatef(0.0,0.0,-15.0);
glRotatef(60.0,0.0,0.0,1.0);
glutWireTeapot(5.0);
```



Transformações em OpenGL

O comando de rotação glRotatef(A,p,q,r) rotaciona cada ponto de um objeto segundo um eixo ao longo a linha desde a origem O=(0,0,0) ao ponto (p,q,r). O ângulo de rotação é A graus, medido em sentido anti-horário quando vemos a origem desde (p,q,r).



Transformações em OpenGL

Experimento: Sucessivamente substitua o comando de rotação pelos seguintes, em cada caso tente deduzir qual será o resultado, antes de rodar o programa.

1. glRotatef(60.0,0.0,0.0,-1.0)
2. glRotatef(-60.0,0.0,0.0,1.0)
3. glRotatef(60.0,1.0,0.0.0.0)
4. glRotatef(60.0,0.0,1.0.0.0)
5. glRotatef(60.0,1.0,0.0.1.0)

Compondo Transformações

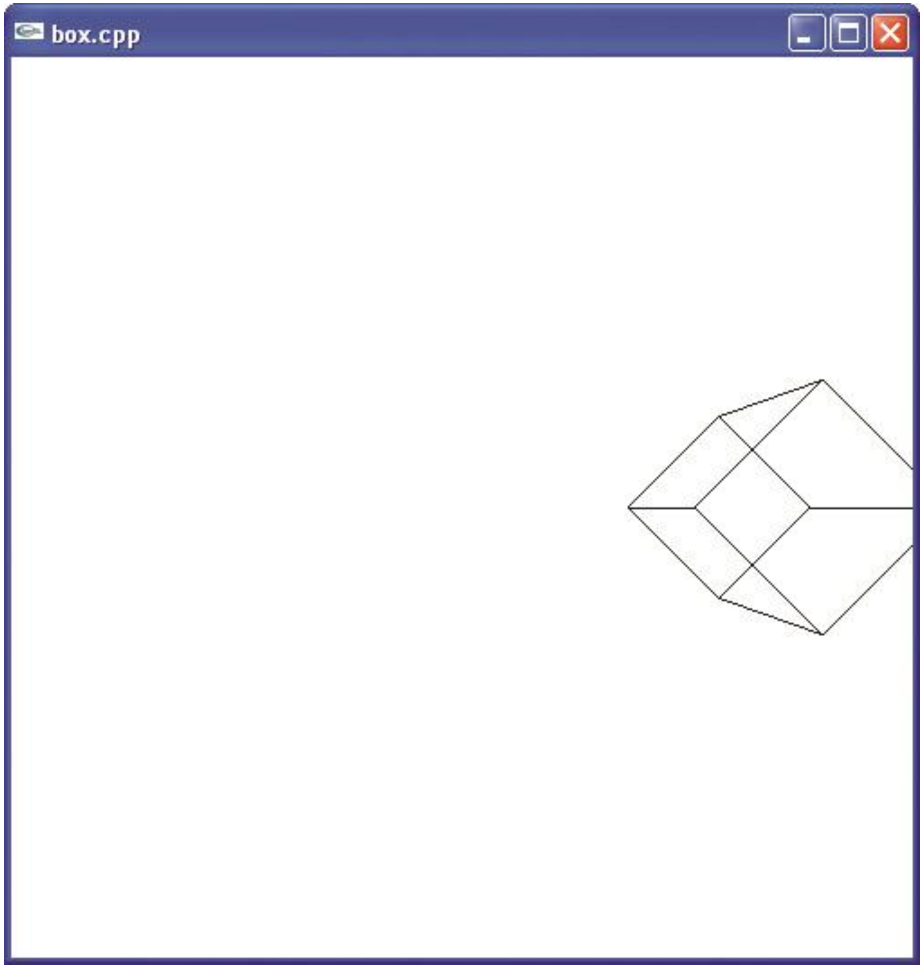
Experimento: Aplique três transformações substituindo o bloco correspondente no programa box.cpp.

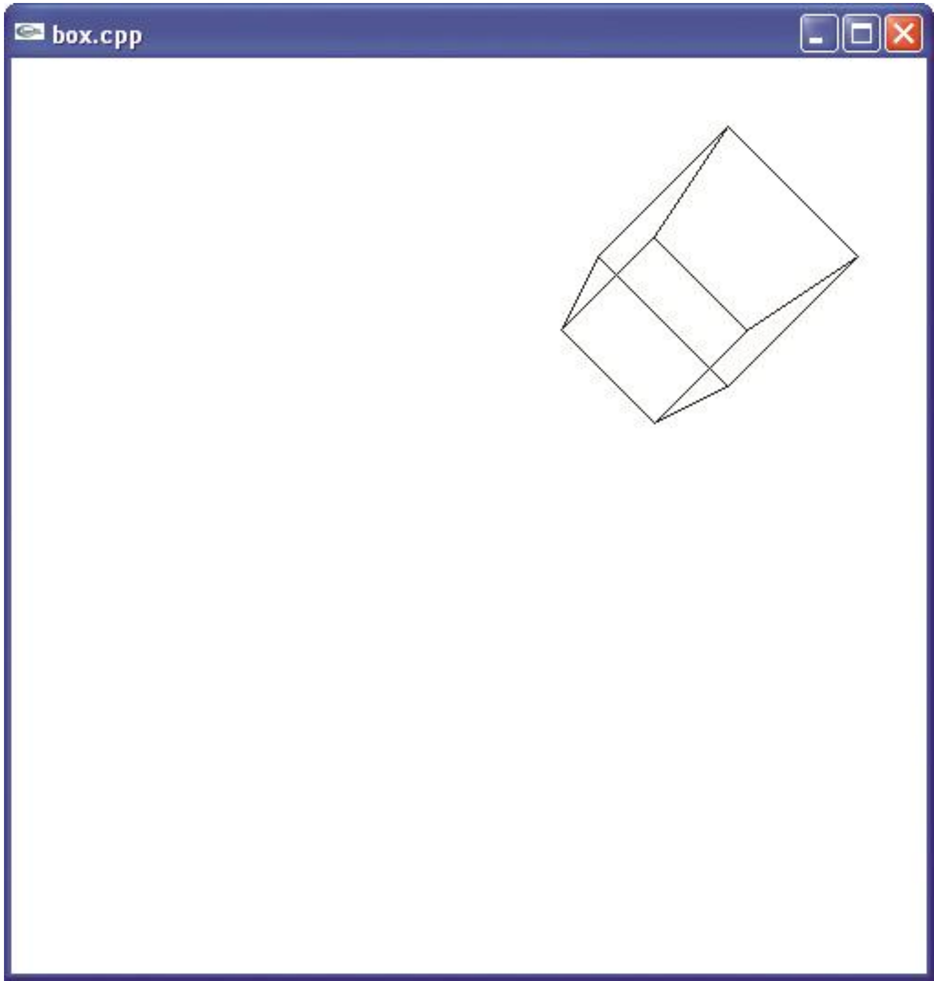
```
//Modeling transformations
glTranslatef(0.0,0.0,-15.0);
glTranslatef(10.0,0.0,0.0);
glRotatef(45.0,0.0,0.0,1.0)
```

A caixa é rotacionada 45 graus ao redor do eixo z e então transladada 10 unidades. A primeira translação (0.0,0.0,-15.0) serve, como já mencionado, para levar a caixa dentro do volume de visualização especificado.

Agora troque as transformações para que a caixa seja primeiro transladada e depois rotacionada.

Compondo Transformações





Compondo Transformações

Exercício: Aplique três transformações, esta vez substituindo o bloco correspondente por:

```
//Modeling transformations
glTranslatef(0.0,0.0,-15.0);
glRotatef(45.0,0.0,0.0,1.0);
glScalef(1.0,3.0,1.0);
```

Troque as transformações de forma que tenhamos:

```
//Modeling transformations
glTranslatef(0.0,0.0,-15.0);
glScalef(1.0,3.0,1.0);
glRotatef(45.0,0.0,0.0,1.0);
```

Diga sua conclusão.

Compondo Transformações

A matriz da composição de duas transformações é o produto de suas matrizes. Generalizando, se aplicarmos sucessivamente as transformações tn,tn-1,...,t1 a um vértice V, então temos.

t1(t2(...tn(V)...))=M1(M2(...(MnV)...))=(M1M2...Mn)V.

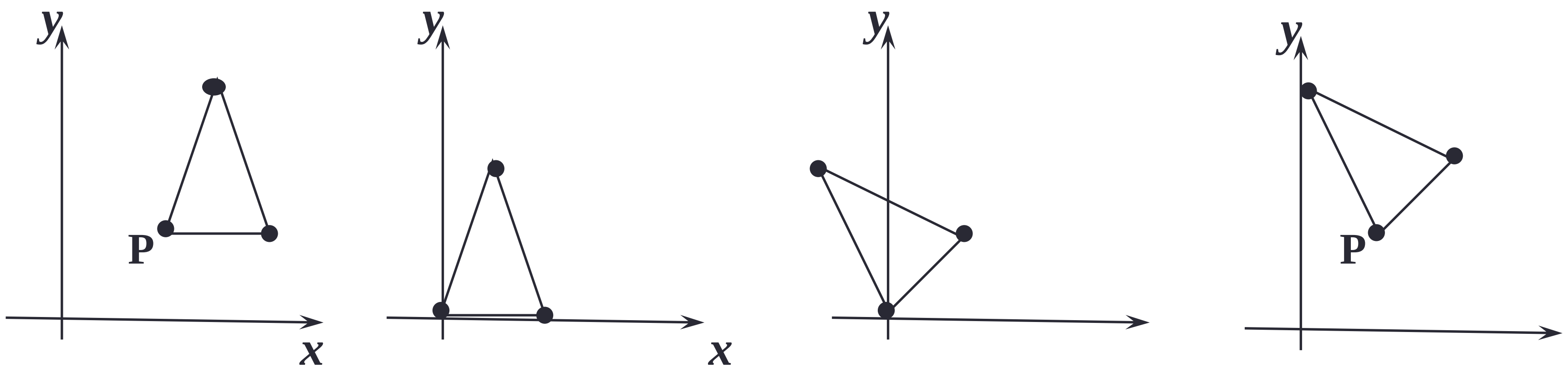
No código

```
                              //M=I, inicialmente
modelingTransformation 1;     //M=IM1 = M1
modelingTransformation 2;     //M=M1M2
...
modelingTransformation n-1;   //M=M1M2...Mn-1
modelingTransformation n;     //M=M1M2...Mn-1Mn
objeto;
```

Rotação em torno de um ponto que não é a origem

Para alterar a orientação de um objeto em torno de um certo ponto, é necessário,
(1) realizar uma translação para localizar esse ponto na origem do sistema,
(2) aplicar a rotação desejada e,
(3) Aplicar uma translação inversa

Rotação em torno de um ponto que não é a origem



(1) (2) (3)

Objeto original	Depois da Translação de P à origem	Após Rotação	Após Translação retorna à posição original

Rotação em torno de um ponto que não é a origem

Faça

glTranslatef (0.0,0.0,-15.0);

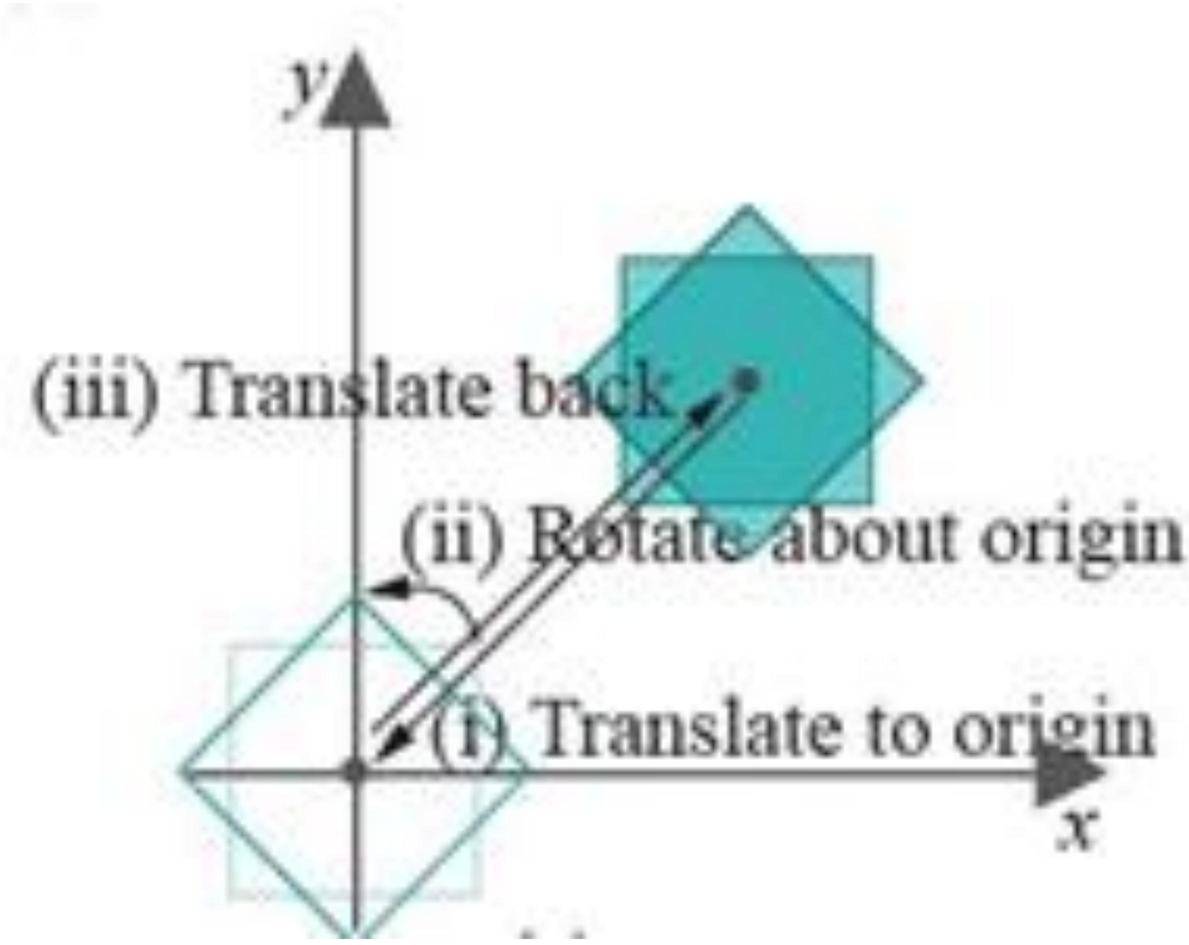
glTranslatef(7.5,7.5,0.0); /*Translade de volta*/

glRotatef(45.0,0.0,0.0,1.0); /*Rotacione em torno da origem */

glTranslatef(-7.5,-7.5,0.0); /* Translade ate a origem*/

glRectf(5.0,5.0,10.0,10.0);

Rotação em torno de um ponto que não é a origem



Posicionando múltiplos objetos

Substitua a rotina de desenho do box.cpp original pelo seguinte trecho:

```
void drawScene(void)
{glClear (GL_COLOR_BUFFER_BIT);
glColor 3f(0.0,0.0,0.0);
glLoadIdentity();
//Modeling transformations
glTranslatef(0.0,0.0,-15.0);
//glRotatef(45.0,0.0,0.0,1.0);
glTranslatef(5.0,0.0,0.0);
```

Posicionando múltiplos objetos

```
glutWireCube(5.0); //Box
//More modeling transformations
glTranslatef(0.0,1.0,0.0);
glutWireSphere(2.0,10,8); //Sphere
glFlush();
}
```

Observe o resultado e compare com o resultado após descomentar o glRotatef.

Compondo transformações

Experimento: Rode composeTransformations.cpp. Veja o efeito da composição de transformações, pressionando a tecla up sucessivamente.

Pilha de Matrizes (Modelview)

OpenGL mantém três tipos diferentes de pilhas de matrizes: modelview, projection e texture.

- glMatrixMode(GL_MODELVIEW);
  - Define a matriz de transformação de visualização. Após isso deve-se definir as transformações geométricas glRotate e/ou glTranslate para orientar e posicionar os objetos em relação da câmera (O comando simplificado gluLookAt pode também ser usado como será visto posteriormente).

Pilha de Matrizes (Modelview)

Experimento: Deseja-se criar um personagem. Inicia-se com o tronco aproximado por um cubo alongado e posiciona-se uma esfera sobre o topo do cube para simular a cabeça. Substitua o rotina de desenho do programa box.cpp pelo seguinte trecho:

Pilha de Matrizes (Modelview)

```
Void drawScene (void)
{glClear (GL_COLOR_BUFFER_BIT);
glColor 3f(0.0,0.0,0.0);
glLoadIdentity();
glTranslatef(0.0,0.0,-15.0);
glScalef(1.0,2.0,1.0);
glutWireCube(5.0);
glTranslatef(0.0,7.0,0.0);
glutWireSphere(2.0,10,8);
glFlush();
```

Pilha de Matrizes (Modelview)

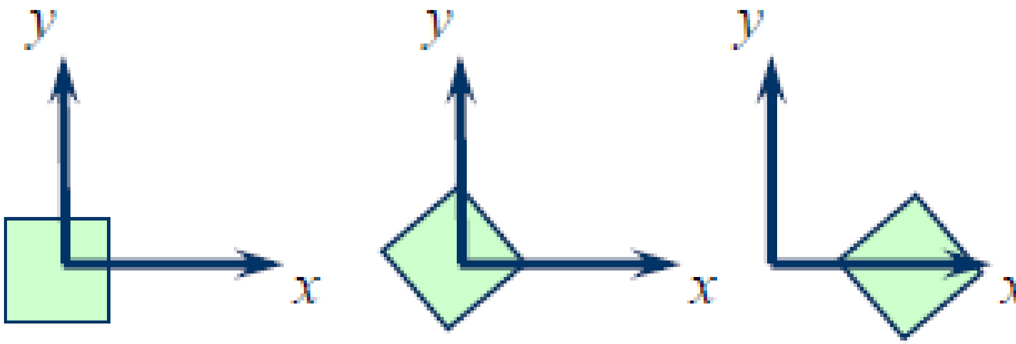
O que você obteve como resultado e por que?

Como pensar nas rotações

1. Considerar um sistema coordenado global fixo.
   - Você tera que pensar que as transformações ocorrem na ordem inversa da que aparecem no código.

glTranslatef(5.0,0.0,0.0)

glRotatef(45,0.0,0.0,1.0)



Como pensar nas rotações

1. Considerar um sistema coordenado global fixo.
   - Dependendo do caso, às vezes pensar na ordem inversa pode se tornar confuso.
   - Há uma forma alternativa de pensar nas rotações.

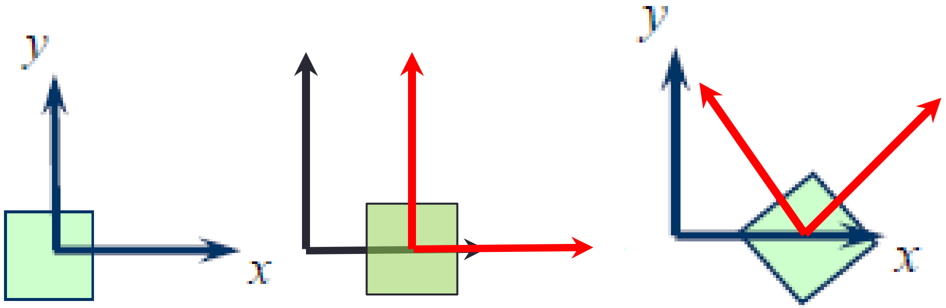
Como pensar nas rotações

2. Considerar um sistema coordenado local.

- Outro sistema é o sistema local móvel associado ao objeto, que faz uso de uma ordem natural das transformações.
- Neste caso, o sistema de coordenadas é fixo ao objeto da cena. Todas as operações são relativas ao novo sistema de coordenadas

glTranslatef(5.0,0.0,0.0)

glRotatef(45,0.0,0.0,1.0)



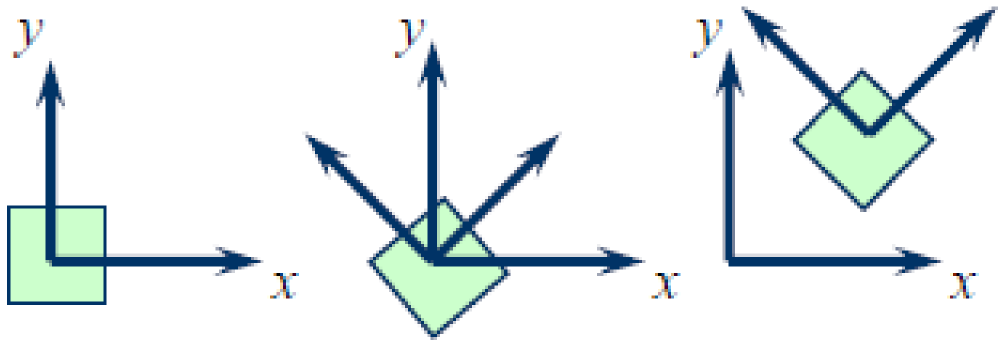
Como pensar nas rotações

2. Considerar um sistema coordenado local.

- E se invertermos a ordem teremos:

glRotatef(45,0.0,0.0,1.0)

glTranslatef(5.0,0.0,0.0)



Pilha de Matrizes – Hierarquia de objetos

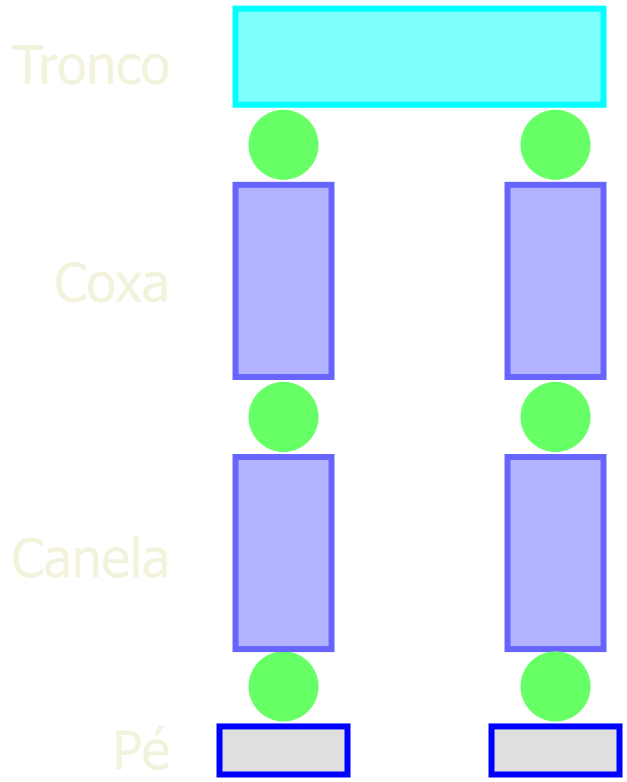
As vezes queremos construir objetos hierarquicos nos quais objetos complicados são construidos a partir de objetos mais simples. Por exemplo,

(a) Uma mesa ou

(b) um automovel com 4 rodas onde cada uma delas é ligada ao carro com cinco parafusos.

(c) O corpo humano

Pilha de Matrizes – Hierarquia de objetos



Pilha de Matrizes – Hierarquia de objetos

Os passos para desenhar um carro serião:

-Desenhe o corpo do carro.

-Guarde a posição onde estamos e translade à direita a roda da frente.

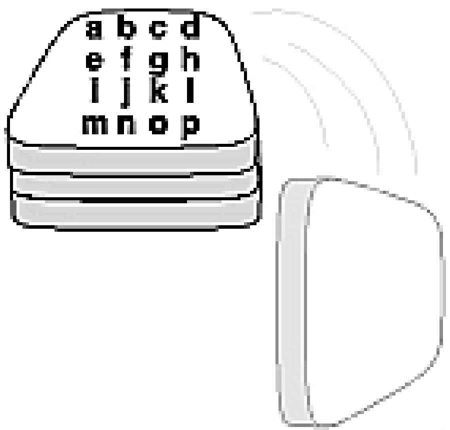
-Desenhe a roda e elimine a última translação talque a posição corrente esteja de volta na origem do carro.

-Guarde a posição onde estamos e translade à esquerda a roda da frente ....

Assim, para cada roda, desenhamos a roda, guardamos a posição onde estamos, e sucessivamente transladamos a cada uma das posições que os parafusos são desenhados, eliminamos as transformações depois que cada parafuso é desenhado.

Pilha de Matrizes – Hierarquia de objetos

glPushMatrix



glPopMatrix

Pilha de Matrizes – Hierarquia de objetos

Desenhe um automovel asumindo que existem as rotinas que desenham o corpo do carro, a roda e o parafuso.

Example 3-4 : Pushing and Popping the Matrix

```
draw_wheel_and_bolts(){
  long i;
  draw_wheel();
  for(i=0;i<5;i++){
     glPushMatrix();
     glRotatef(72.0*i,0.0,0.0,1.0);
     glTranslatef(3.0,0.0,0.0);
     draw_bolt();
     glPopMatrix();
  }}
```

Pilha de Matrizes – Hierarquia de objetos

```
draw_body_and_wheel_and_bolts(){
  draw_car_body();
  glPushMatrix();
  glTranslatef(40,0,30); /*move to first wheel position*/
  draw_wheel_and_bolts();
  glPopMatrix();
  glPushMatrix();
  glTranslatef(40,0,-30); /*move to 2nd wheel position*/
  draw_wheel_and_bolts();
  glPopMatrix();
... /*draw last two wheels similarly*/
}
```

Orientar a câmera em direção da cena (transformação de visualização)

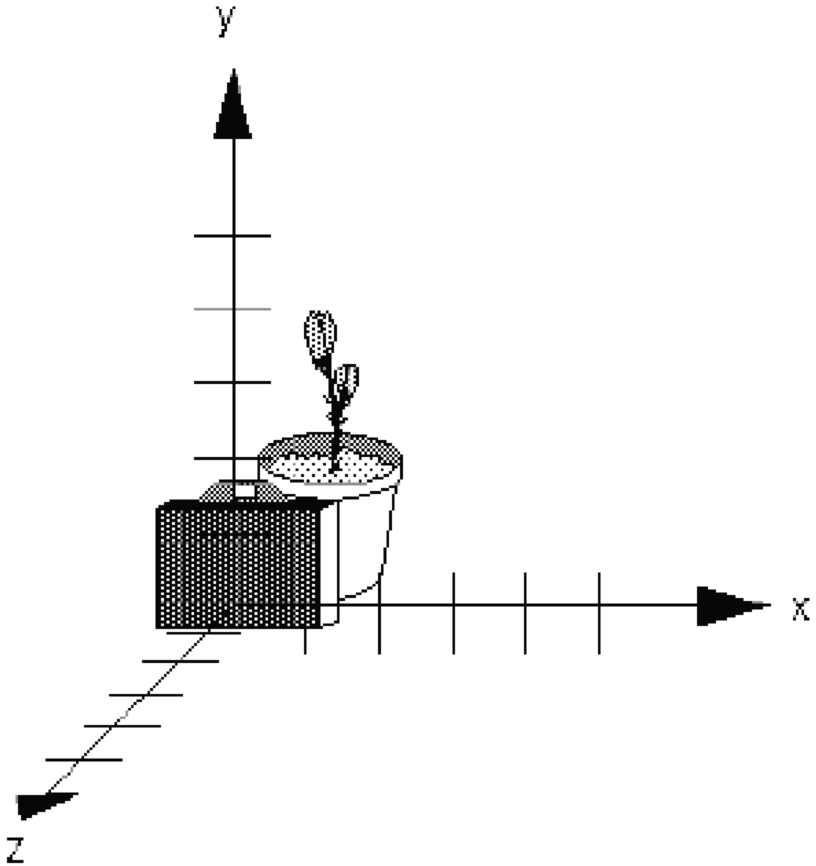
A câmera em OpenGL “por default” tem sua posição na origem de coordenadas (0,0,0) e a sua orientação é com vetor up=(0,1,0). Existem duas opções para mudar sua posição e orientação:

(1) Usar glTranslate*() e glRotate*(). Move a camera ou move todos os objetos em relação a uma camera fixa;

(2) gluLookAt()

Visualizando devidamente o objeto (Exemplo)

- Objeto e câmera na origem



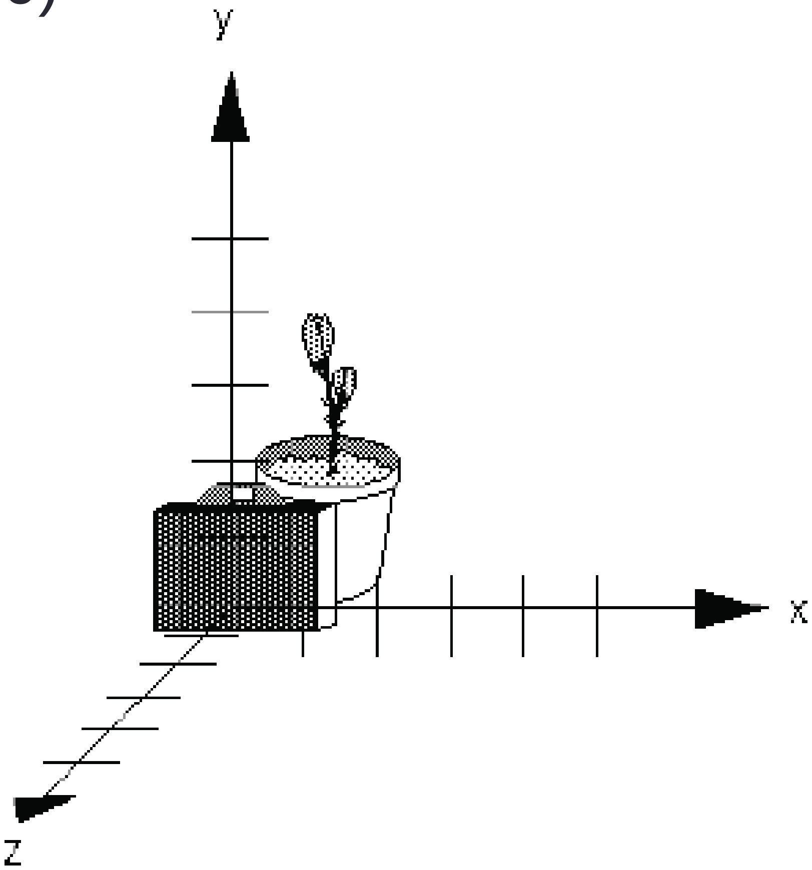
Visualizando devidamente o objeto

Com a câmera na origem (0,0,0) não posso visualizar devidamente um objeto na posição (0,0,0)

Para visualizá-lo tenho duas opções:

(a) Mudar a câmera, ou

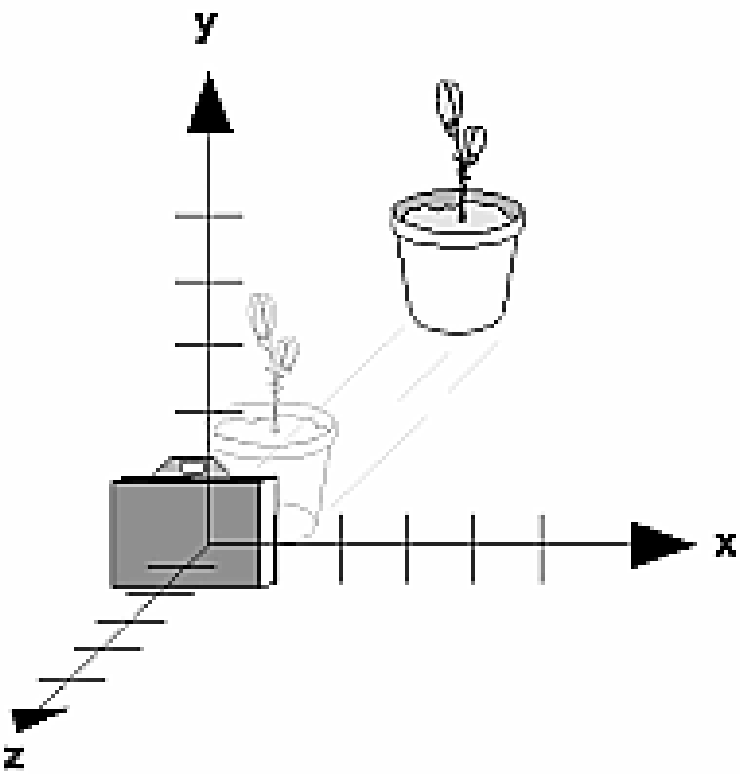
(b) Mudar o objeto



Usando glTranslate() e glRotate()

(b) Mudando o objeto

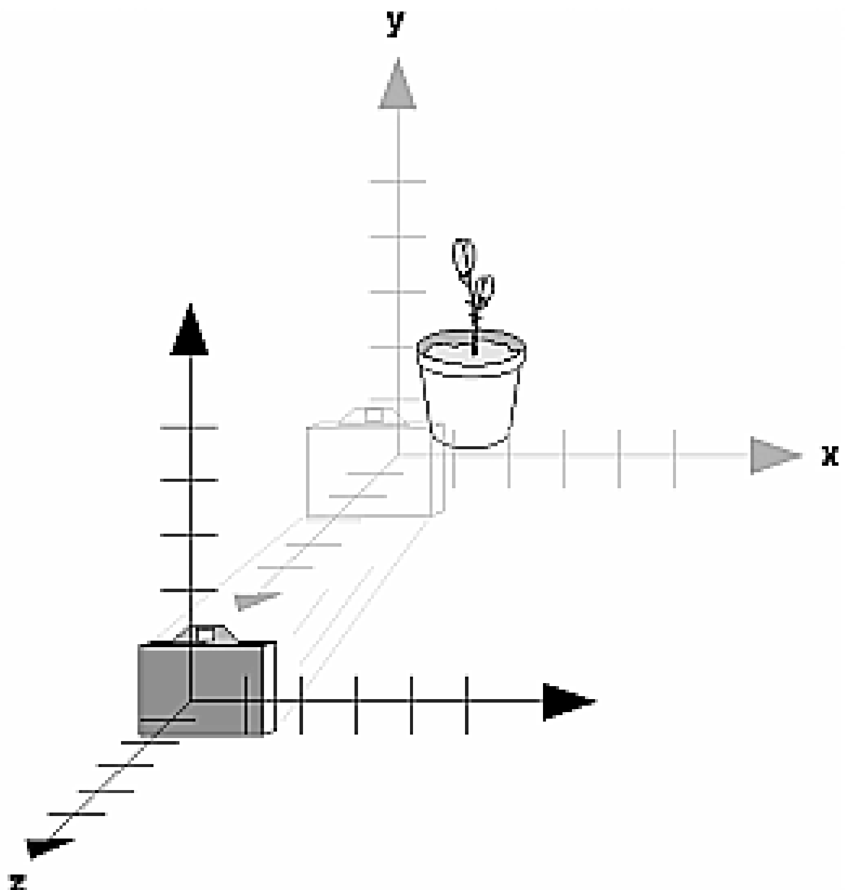
glTranslatef(0.0, 0.0, -5.0);



Usando gluLookAt

(a) Mudando a câmera

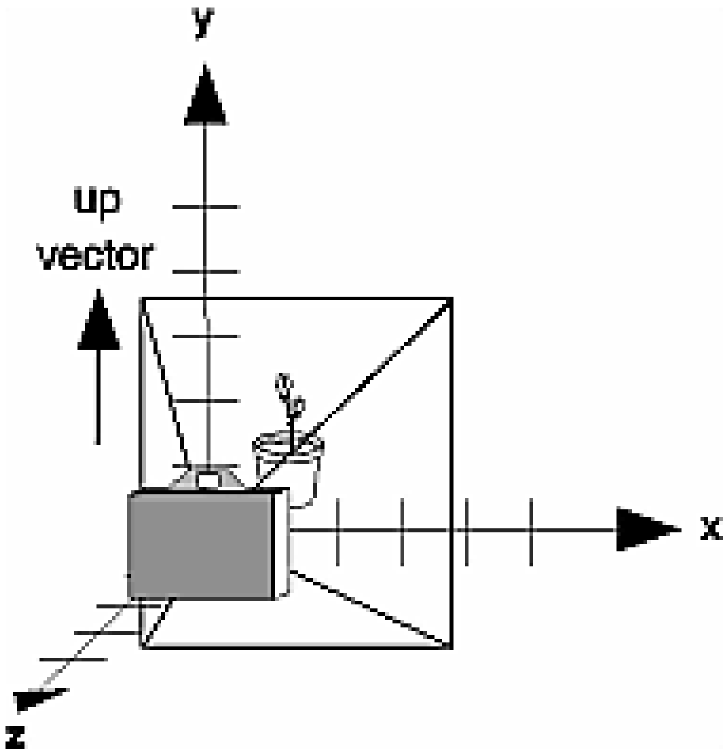
gluLookAt(eyex, eyey, eyez,
centerx, centery, centerz,
upx, upy, upz)

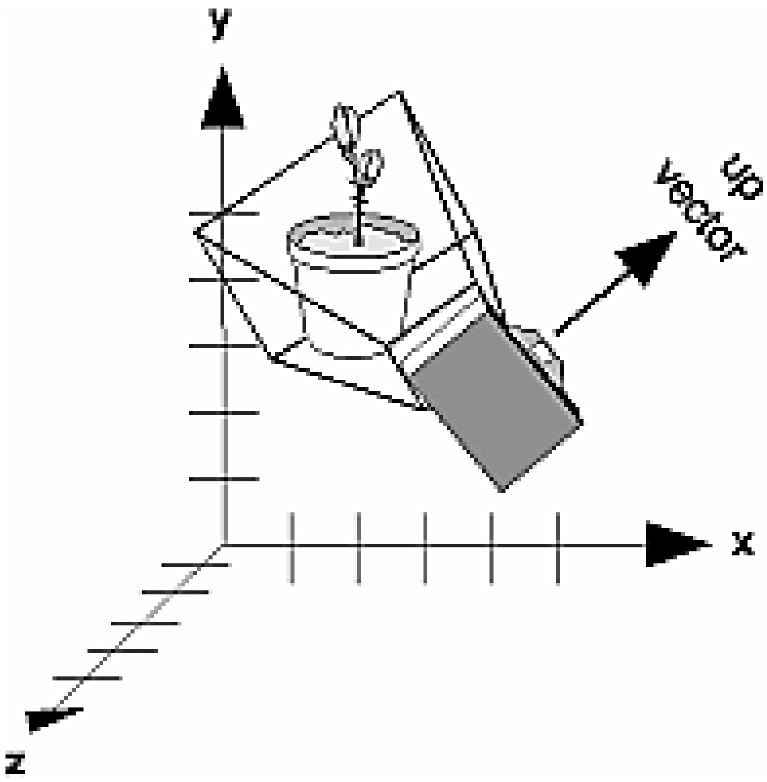


gluLookAt

- A cena é construída na origem e definimos uma posição arbitrária para a câmera
- void gluLookAt (eyex, eyey, eyez,
  centerx, centery, centerz,
  upx, upy, upz);
  - Eye: localização da camera
  - Center: para onde a camera aponta
  - Up: vetor de direção de topo da camera

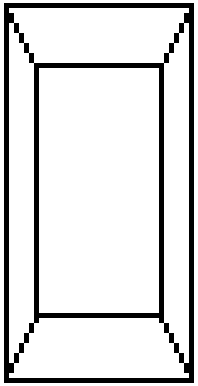
gluLookAt





Exemplo – Cubo (Programa cube.c)

Um cubo é escalado pela transformação de modelagem glScalef (1.0, 2.0, 1.0). A transformação de visualização **gluLookAt()**, posiciona e orienta a câmera em direção do cubo. As transformações de projeção e viewport são também especificadas.



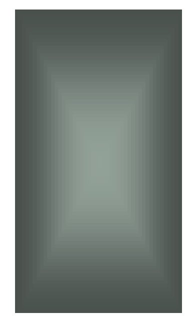
Exercício

(1) Faça um programa C/OpenGL que desenhe uma mesa retangular, a partir de cubos (glutWireCube) e transformações de modelagem.

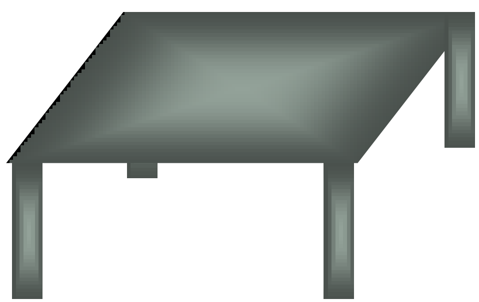
(2) Oriente devidamente a câmera, de forma que obtenhamos as seguintes imagens da mesa:

(a)

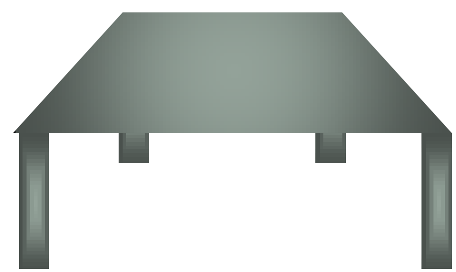
(b)



(c)

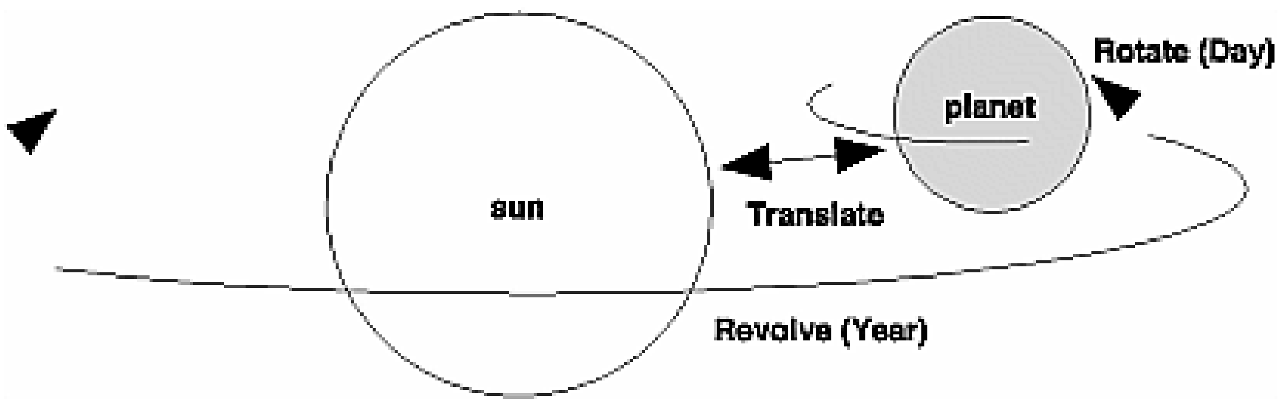


(d)



Exercício

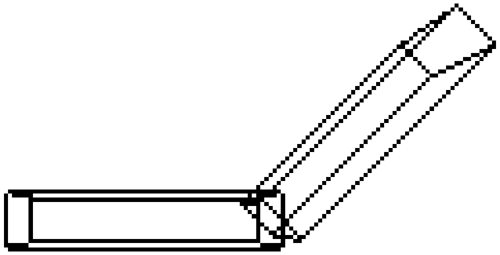
(3) O programa planet.c usa **glRotate*()** para rotacionar um planeta ao redor do sol e para rotacionar o planeta ao redor do seu próprio eixo.



- Modifique o programa para que acrescente mais dois planetas com seus respectivos satélites. Como se trata de objetos hierárquicos use glPushMatrix e glPopMatrix (vide aula).

Exercício

(4) O programa robot.c constrói o braço articulado de um robô usando dois "cubos alongados". O robô possui articulações no ombro e no cotovelo.



- Modifique o programa para que acrescente a mão e dedos.

Exercício

(5) Seguindo as orientações dadas faça um programa que desenhe um carro com cinco parafusos em cada uma das suas quatro rodas.