

COMPUTAÇÃO GRÁFICA – OPENGL

Profa. Mercedes Gonzales
Márquez

Curvas de Bézier - Algoritmo

Exercício:

- (1) O programa superficies.cpp permite o ingresso interativo (pelo cliques do mouse) de $n+1$ pontos de controle e constrói a curva de Bézier de grau n , correspondente. Responda as seguintes questões:
 - a. Qual trecho de código constrói as funções de Bernstein (Jni)
 - b. Qual trecho de código constrói a curva de Bézier.
 - c. Em qual trecho de código a curva é desenhada
 - d. O programa funciona, mas pode ser melhorado. Identifique quais melhoras podem ser feitas.

Curvas de Bézier - Algoritmo

Exercício:

(2) Rode e entenda os programas [bezierCurves.cpp](#) e [bezierCurveWithEvalMesh.cpp](#) que desenharam curvas de Bézier usando comandos de OpenGL. Responda as seguintes questões:

- a. Qual trecho de código constrói as funções de Bernstein (Jni)
- b. Qual trecho de código constrói a curva de Bézier.
- c. Em qual trecho de código a curva é desenhada
- d. O programa apresenta uma proposta de interação, você teria outra proposta.

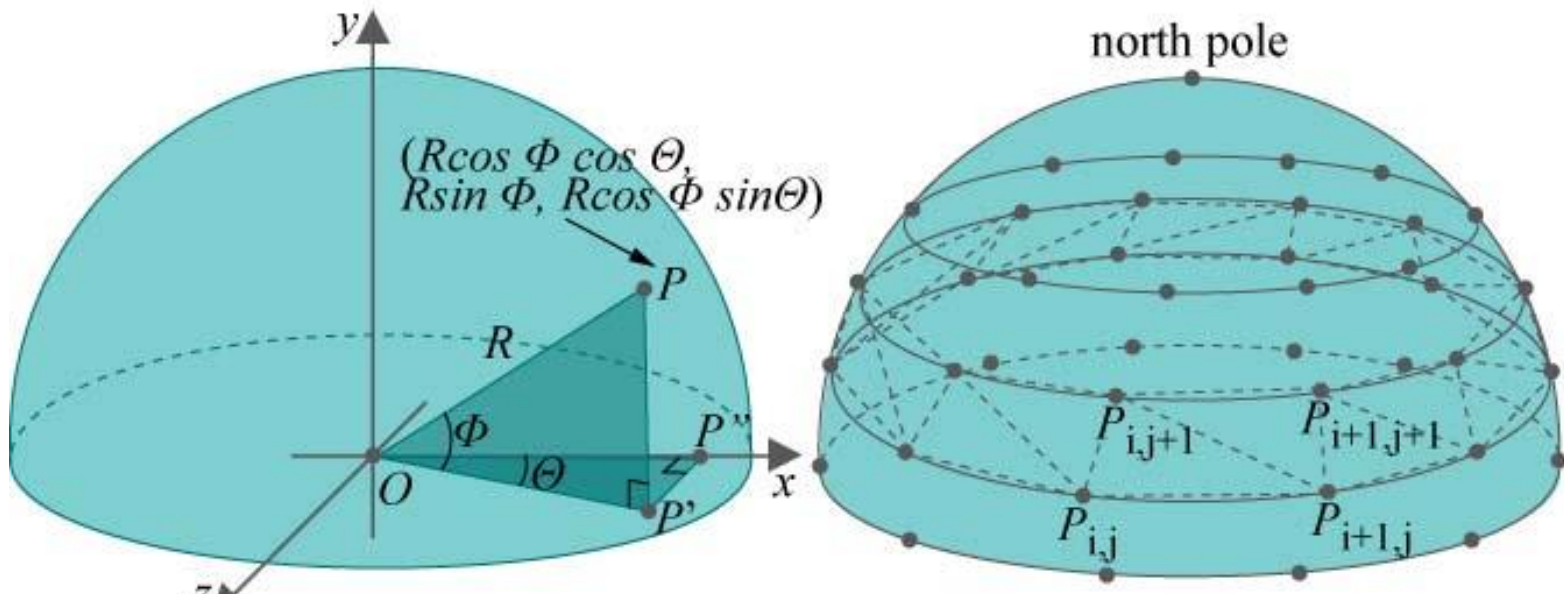
Aproximando Objetos Bi-dimensionais

Considere um hemisfério de raio R , centralizado na origem O e com sua base circular sobre o plano xz . Suponha que as coordenadas esféricas de um ponto P sobre o hemisfério são a longitude θ e a latitude ϕ .

As coordenadas cartesianas são:

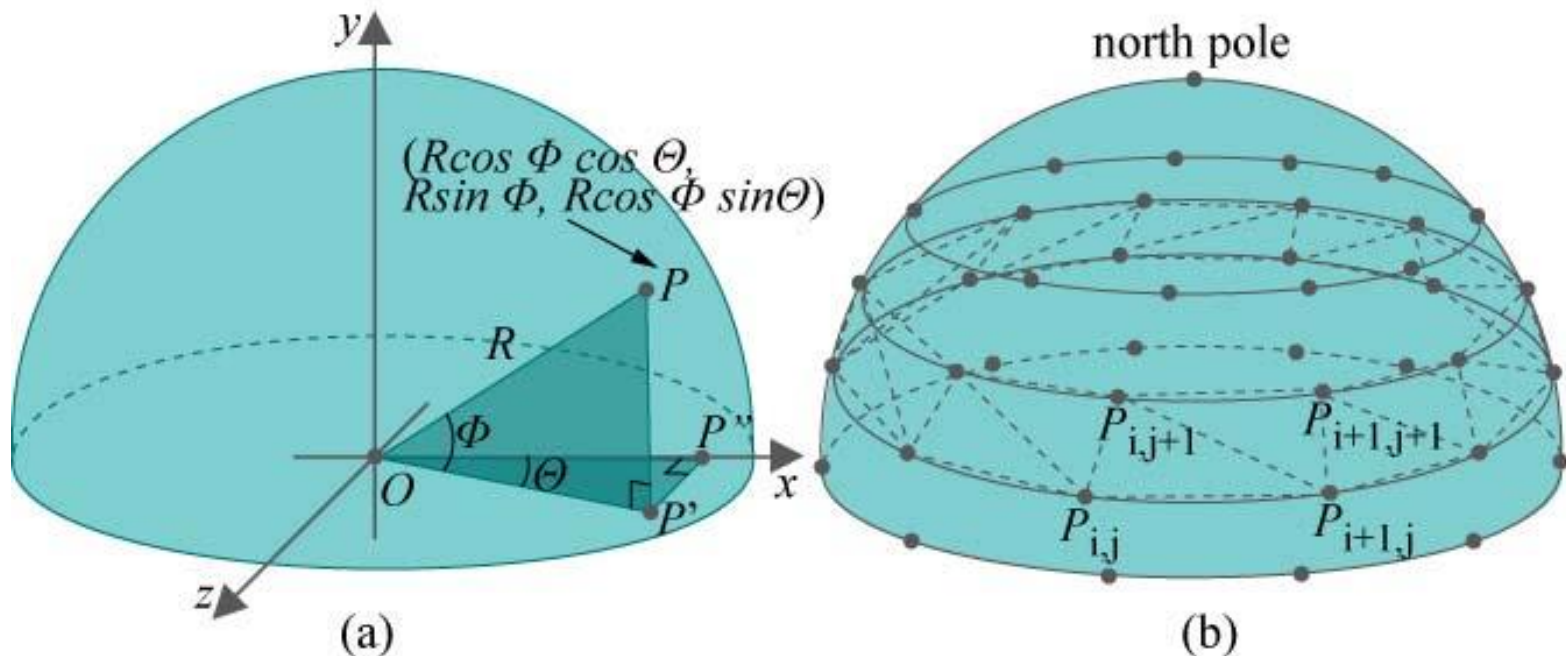
$$(R \cos \phi \cos \theta, R \sin \phi, R \cos \phi \sin \theta),$$

$$0 \leq \theta \leq 2\pi \text{ e } 0 \leq \phi \leq \pi/2$$



Aproximando Objetos Bi-dimensionais

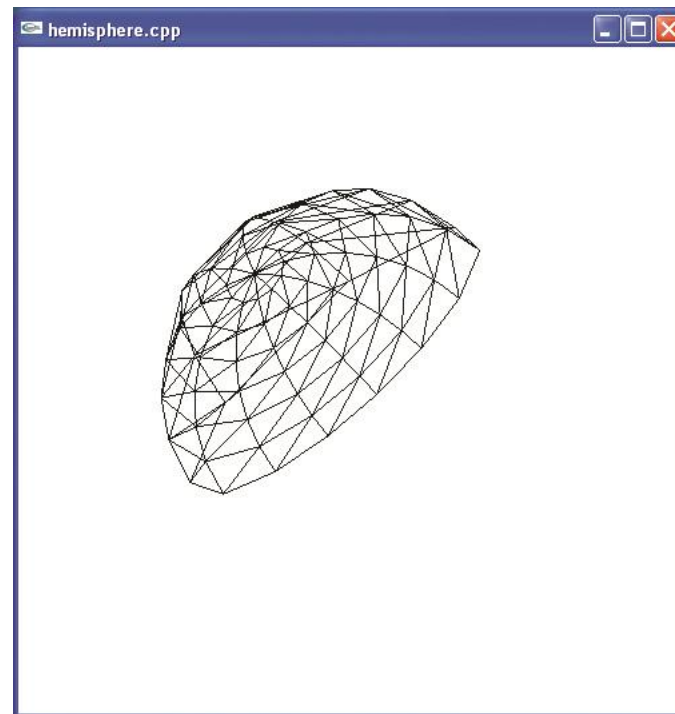
Amostramos o hemisfério em uma malha de $(p+1)(q+1)$ pontos P_{ij} , $0 \leq i \leq p$, $0 \leq j \leq q$, onde a longitude de P_{ij} é $(i/p) \cdot 2\pi$ e sua latitude $(j/q) \cdot \pi/2$. Em outras palavras $(p+1)$ pontos longitudinalmente igualmente espaçados são escolhidos entre cada uma das $q+1$ latitudes igualmente espaçadas. Na figura $p=10$ e $q=4$.



Aproximando Objetos Bi-dimensionais

Experimento 2.26. Rode hemisphere.cpp que implementa exatamente a estratégia descrita.

Experimento 2.27.



Aproximando Objetos Bi-dimensionais

even now as the syntax is fairly intuitive. The set of three `glRotatef()`s, particularly, comes in handy to re-align a scene.

Exercise 2.31. (Programming) Modify `hemisphere.cpp` to draw:

- (a) the bottom half of a hemisphere (Figure 2.35(a)).
- (b) a 30° slice of a hemisphere (Figure 2.35(b)).

Make sure the 'P/p/Q/q' keys still work.

Exercise 2.32. (Programming) Just to get you thinking about animation, which we'll be studying in depth soon enough, guess the effect of replacing `glTranslatef(0.0, 0.0, -10.0)` with `glTranslatef(0.0, 0.0, -20.0)` in `hemisphere.cpp`. Verify.

And, here are some more things to draw.

Exercise 2.33. (Programming) Draw the objects shown in Figure 2.36. Give the user an option to toggle between filled and wireframe renderings.

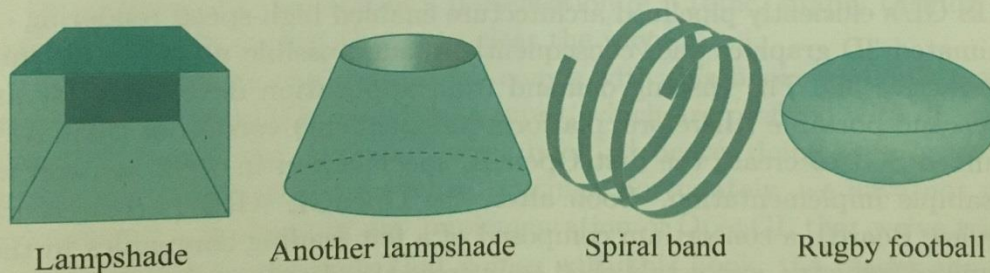


Figure 2.36: More things to draw.

A suggestion for the football, or ellipsoid, is to modify `hemisphere.cpp` to make half of an ellipsoid (a hemi-ellipsoid?). Two hemi-ellipsoids back to back would then give a whole ellipsoid.

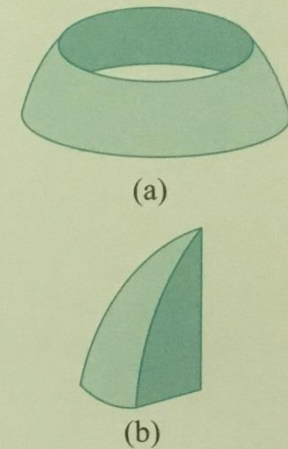


Figure 2.35: (a) Half a hemisphere (b) Slice of a hemisphere.

Curvas e Superfícies Bézier em OpenGL

1. Leia e entenda os programas [bezierSurface.cpp](#) e [bezierCanoe.cpp](#) que desenharam superfícies de Bézier usando comandos de OpenGL. Explique os comandos que geram as superfícies em ambos programas.

Quádricas da GLU

A biblioteca GLU provê a renderização automática de objetos tridimensionais como [esferas](#), [cilindros](#) e [discos](#).

- Esferas:

```
gluSphere(GLUquadricObj *obj, GLdouble radius, GLint slices, GLint stacks)
```

- Cilindros:

```
gluCylinder(GLUquadricObj *obj, GLdouble baseRadius, GLdouble topRadius,  
GLdouble height, GLint slices, GLint stacks)
```

`topRadius == zero`, permite criar cone.

- Discos:

```
gluDisk(GLUquadricObj *obj, GLdouble innerRadius, GLdouble outerRadius,  
GLint slices, GLint loops)
```

`innerRadius != zero`, permite criar discos com furos.

Veja e rode o programa `gluQuadrics.cpp` da pasta `SumantaGuha`

Modelagem usando Superfícies Bézier e primitivas

1. Leia e entenda o programa torpedo.cpp que desenha um torpedo composto de diferentes pedaços:
 - (i) Corpo: cilindro da GLU
 - (ii) Nariz: hemisferio
 - (iii) Três barbatanas: discos parciais da GLU
 - (iv) Disco traseiro : disco da GLU
 - (v) Haste da hélice: cilindro da GLU
 - (vi) Três pás da hélice: pedaços bicúbicos Bezier
2. Desenhe um avião composto de vários pedaços. Use superfícies de Bézier e quádricas.

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 em OpenGL

Experimento 4.3: 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 4.4: 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)`

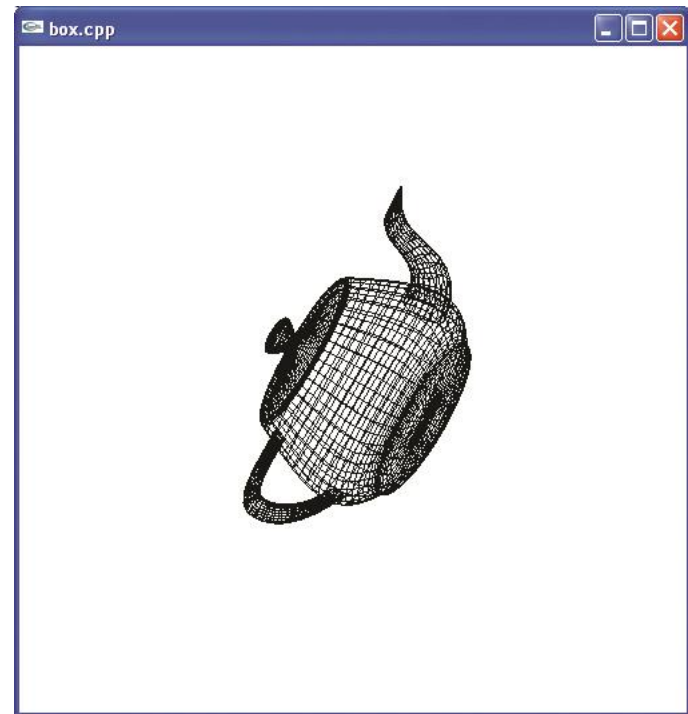
Exercício: A transformação $(x,y,z) \rightarrow (-x,y,z)$ é uma reflexão (espelhamento) em relação ao plano yz.

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

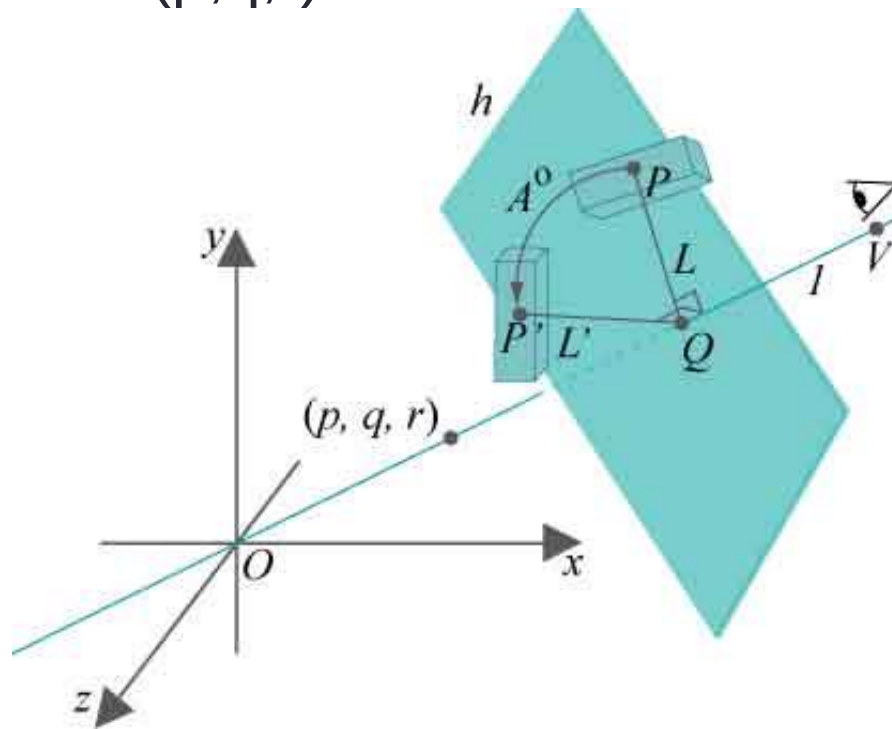
Experimento 4.6: Troque o comando de escala pelo seguinte comando de rotação em box.cpp:

```
//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 4.7: 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 4.9: Aplique três transformações substituindo o bloco correspondente no programa box.cpp.

```
//Modeling transformations
```

```
glTranslatef(0.0,0.0,-15.0);
```

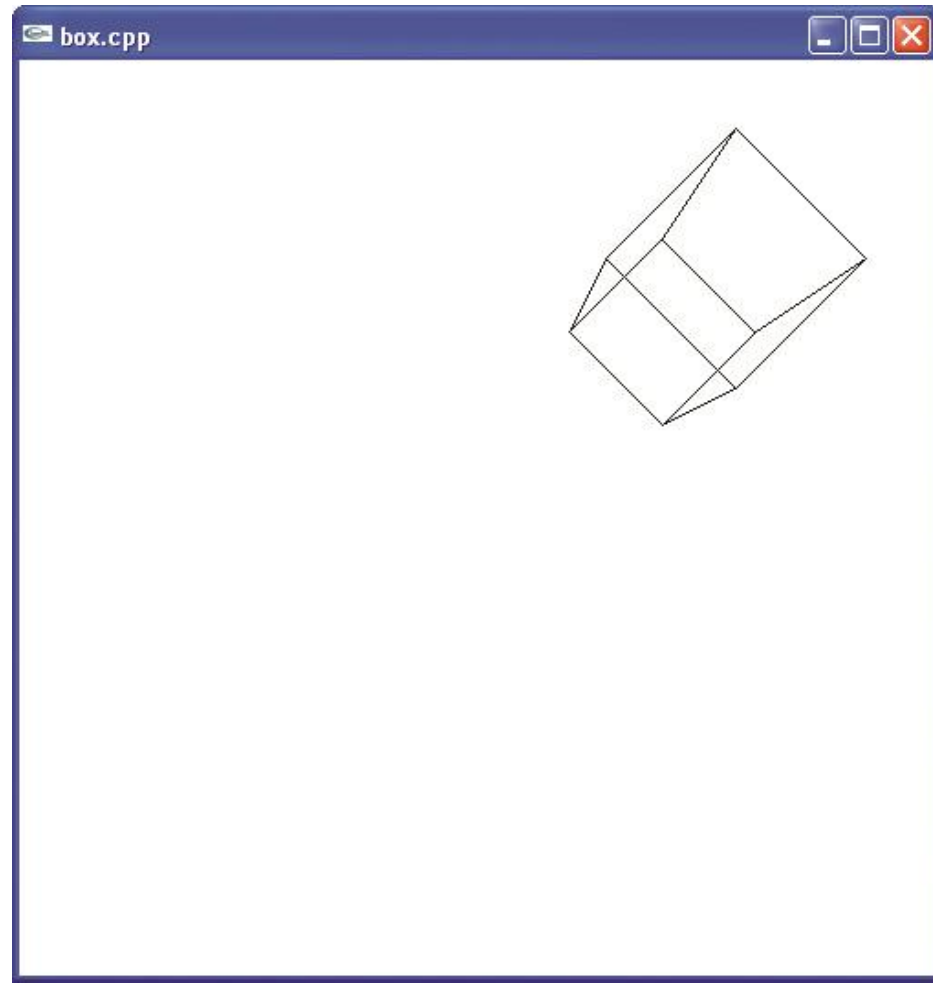
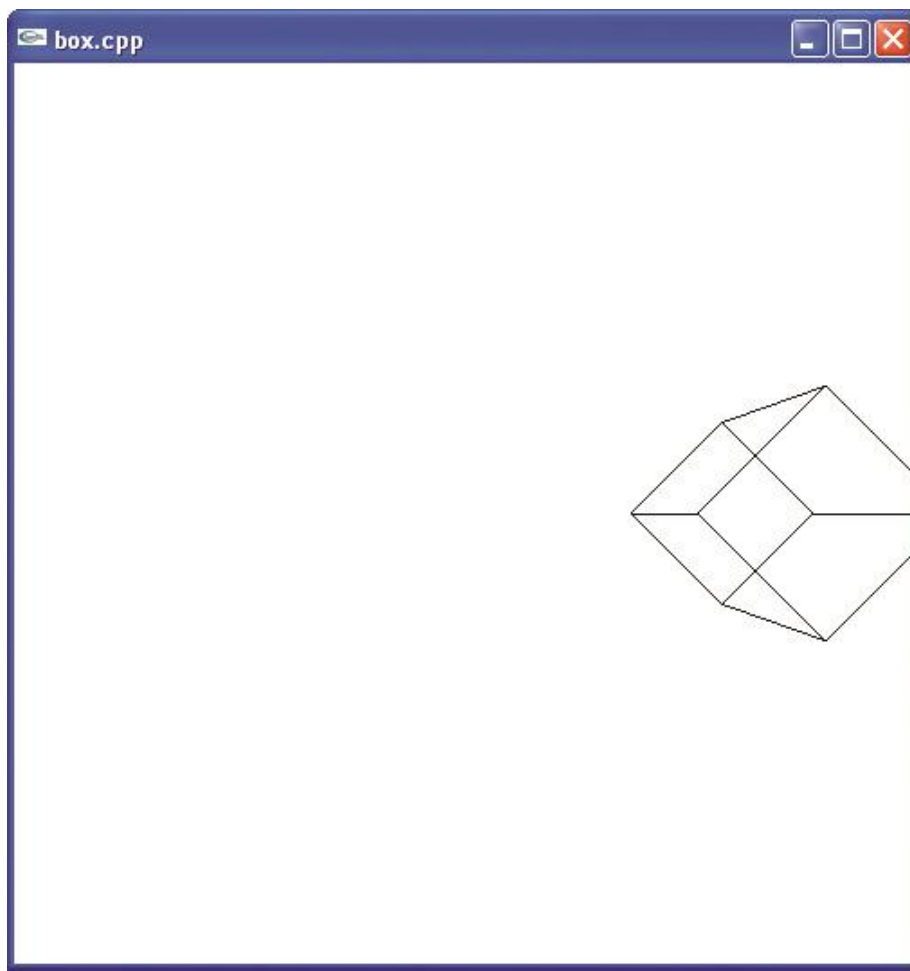
```
glTranslatef(10.0,0.0,0.0);
```

```
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.

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 4.13: 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 4.14: Deseja-se criar um personagem. Inicia-se com o tronco aproximado por um cubo alongado e posiciona-se uma esfera sobre o topo do cubo 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();
```

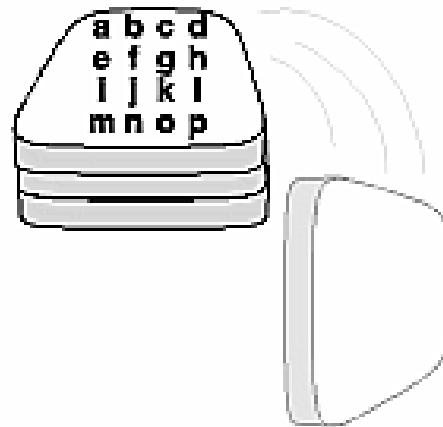
Pilha de Matrizes (Modelview)

O que você obteve como resultado e por que?

Pilha de Matrizes – Hierarquia de objetos



`glPushMatrix`



`glPopMatrix`

Pilha de Matrizes – Hierarquia de objetos

Desenhe um automovel assumindo que existem as rotinas que desenharam 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*/  
}
```

Exercício

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