

Relatório Sobre Transformações em um Objeto Tridimensional

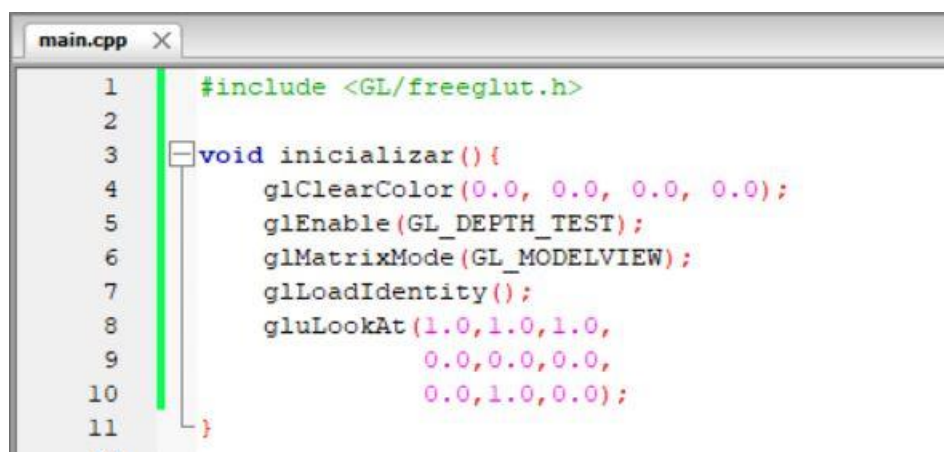
Bruno de Souza Cruz

Resumo

Implementar um programa para aplicar as transformações em um objeto tridimensional, são elas a translação, rotação e escala, utilizando a projeção de perspectiva. O OpenGL é uma API livre utilizada na computação gráfica, para desenvolvimento de aplicativos gráficos ambientes 3D, ela foi utilizada no programa, o GLUT é uma biblioteca de funcionalidades para OpenGL cujo principal objetivo é a abstração do sistema operacional fazendo com que os aplicativos sejam multiplataforma, a ide utilizada foi o Code::Blocks.

1.1. Iniciando no Code::Blocks

Primeiramente a ide foi instalada e o freeglut foi configurado para ser incluído dentro do programa, o método inicializar foi criado, dentro desse método é feito o comando da cor que vai limpar o fundo no caso a cor preto, os parâmetros passados são os valores de RGBA (red, green, blue, alpha), em seguida é habilitado o teste de profundidade pois quando se trabalha com objeto tridimensional, tem que levar em conta na hora de renderizar, decidir então o que ta atrás e o que ta na frente, pra carregar a câmera precisa definir que a matriz é o model view e depois carrega a matriz de identidade, após isso é inserido os parâmetros da câmera, os valores na linha 8 são a posição da câmera, na linha 9 para onde a câmera aponta, e na linha 10 é o vetor view-up.



```
main.cpp X
1  #include <GL/freeglut.h>
2
3  void inicializar(){
4      glClearColor(0.0, 0.0, 0.0, 0.0);
5      glEnable(GL_DEPTH_TEST);
6      glMatrixMode(GL_MODELVIEW);
7      glLoadIdentity();
8      gluLookAt(1.0, 1.0, 1.0,
9               0.0, 0.0, 0.0,
10              0.0, 1.0, 0.0);
11 }
```

O objeto a ser desenhado é um cubo, pra isso foi feito um método, sabe-se que um cubo tem 6 faces, ou seja precisa fazer 6 conjuntos de 4 vértices cada, as vértices são tridimensionais.

```
main.cpp X
12
13 void desenhar_cubo()
14 {
15     glColor3f(1.0,0.0,0.0);
16     glBegin(GL_POLYGON);
17         glVertex3f(-0.5,-0.5,0.5);
18         glVertex3f(0.5,-0.5,0.5);
19         glVertex3f(0.5,0.5,0.5);
20         glVertex3f(-0.5,0.5,0.5);
21     glEnd();
22
23     glColor3f(0.0,1.0,0.0);
24     glBegin(GL_POLYGON);
25         glVertex3f(0.5,0.5,0.5);
26         glVertex3f(0.5,-0.5,0.5);
27         glVertex3f(0.5,-0.5,-0.5);
28         glVertex3f(0.5,0.5,-0.5);
29     glEnd();
30
31     glColor3f(0.0,0.0,1.0);
32     glBegin(GL_POLYGON);
33         glVertex3f(0.5,-0.5,0.5);
34         glVertex3f(-0.5,-0.5,0.5);
35         glVertex3f(-0.5,-0.5,-0.5);
36         glVertex3f(0.5,-0.5,-0.5);
37     glEnd();
38
39     glColor3f(1.0,1.0,0.0);
40     glBegin(GL_POLYGON);
41         glVertex3f(-0.5,0.5,0.5);
42         glVertex3f(0.5,0.5,0.5);
43         glVertex3f(0.5,0.5,-0.5);
44         glVertex3f(-0.5,0.5,-0.5);
45     glEnd();
46
47     glColor3f(0.0,1.0,1.0);
48     glBegin(GL_POLYGON);
49         glVertex3f(-0.5,-0.5,-0.5);
50         glVertex3f(-0.5,0.5,-0.5);
51         glVertex3f(0.5,0.5,-0.5);
52         glVertex3f(0.5,-0.5,-0.5);
53     glEnd();
54
55     glColor3f(1.0,1.0,1.0);
56     glBegin(GL_POLYGON);
57         glVertex3f(-0.5,0.5,-0.5);
58         glVertex3f(-0.5,-0.5,-0.5);
59         glVertex3f(-0.5,-0.5,0.5);
60         glVertex3f(-0.5,0.5,0.5);
61     glEnd();
62 }
63
```

Antes de criar foi definido a cor de cada face as cores escolhidas foram: vermelho (parte positiva do eixo z), verde (parte positiva do eixo x), azul (parte negativa do eixo y), amarelo (parte positiva do eixo y), azul marinho (parte negativa do eixo z), branco (parte negativa do eixo z), as cores são RGB (red, green, blue). Os parâmetros que são passados nas vértices são o (x, y, z).

Após criado os métodos inicializar e o de desenhar o cubo, agora vai ser criado o método de renderizar, no início desse método ele vai limpar o buffer de cor e o buffer de profundidade, quando se trabalha com projeção, janela de recorte, volume de visualização se utiliza o GL_PROJECTION, então é carregado a gl_projection e depois a identidade, a glOrtho quer dizer que vai ser uma projeção ortogonal, os 4 primeiros parâmetros são a janela de recorte ou seja é na câmera, e os 2 últimos se refere ao ponto mais próximo e o ponto mais distante, agora é chamada do método, e por fim acontece a troca do buffer de visualização.

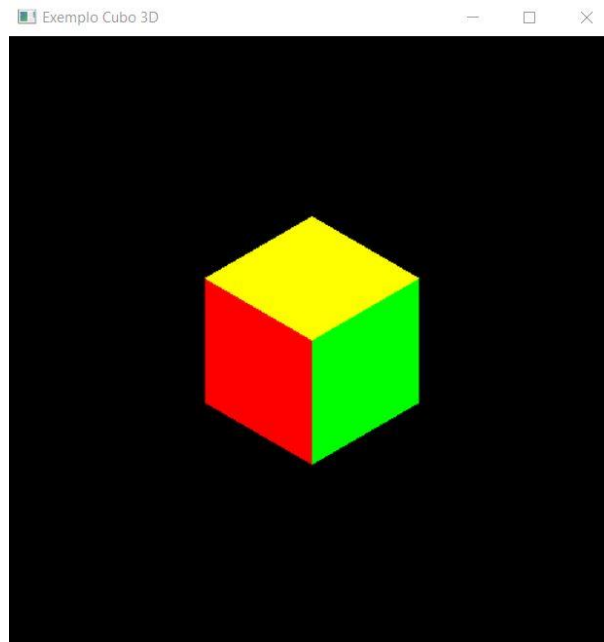
```
63
64 void renderizar()
65 {
66     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
67     glMatrixMode(GL_PROJECTION);
68     glLoadIdentity();
69     glOrtho(-2.0, 2.0, -2.0, 2.0, 0.0, 3.0);
70     desenhar_cubo();
71     glutSwapBuffers();
72 }
73
```

No método principal acontece o gerenciamento de janela, após isso é definido o número de buffer que vai se usar, o sistema de cor, e o buffer de profundidade, é definido o tamanho da janela e a posição do desktop, depois é inserido o título, é definido uma função de call-back pra visualizar o desenho, e por fim tem a chamada da função inicializar.

```
73
74 int main(int argc, char *argv[])
75 {
76     glutInit(&argc, argv);
77     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
78     glutInitWindowSize(500, 500);
79     glutInitWindowPosition(100, 100);
80     glutCreateWindow("Exemplo Cubo 3D");
81     glutDisplayFunc(renderizar);
82     inicializar();
83     glutMainLoop();
84     return 0;
85 }
86
```

1.2 Resultados Iniciais

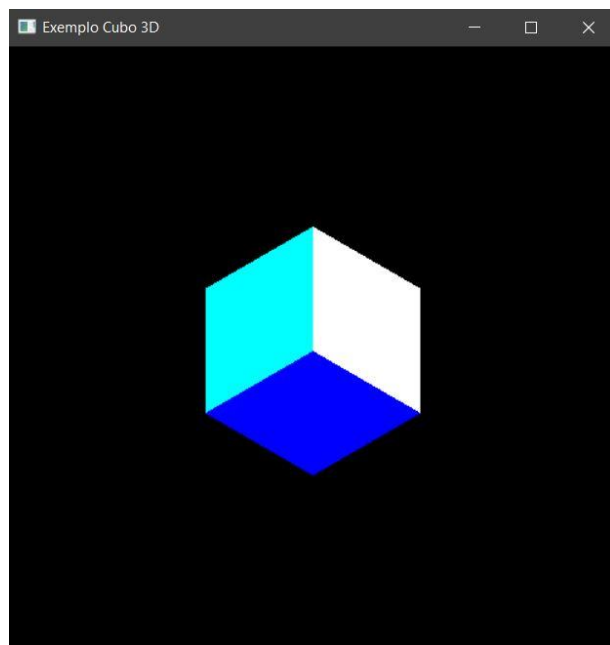
O resultado de todo código acima se vê na (Figura 1.0).



(Figura 1.0)

Lembrando que verde (parte positiva do eixo x), o vermelho (parte positiva do eixo z) e o amarelo (parte positiva do eixo y). A visualização do cubo está dessa forma pois foi definido antes a posição da câmera, ou seja, para ver as faces que estão atrás basta modificar a câmera nas posições (x, y, z), veja o resultado na (Figura 1.1).

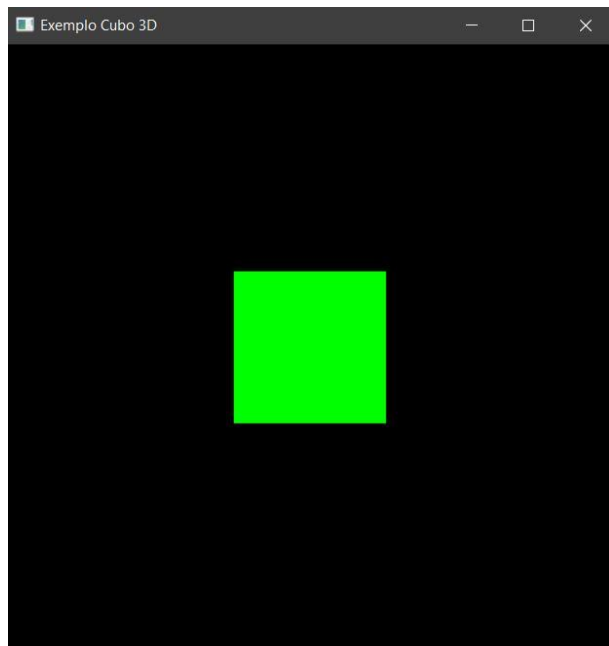
```
gluLookAt (-1.0, -1.0, -1.0,  
           0.0, 0.0, 0.0,  
           0.0, 1.0, 0.0);
```



(Figura 1.1)

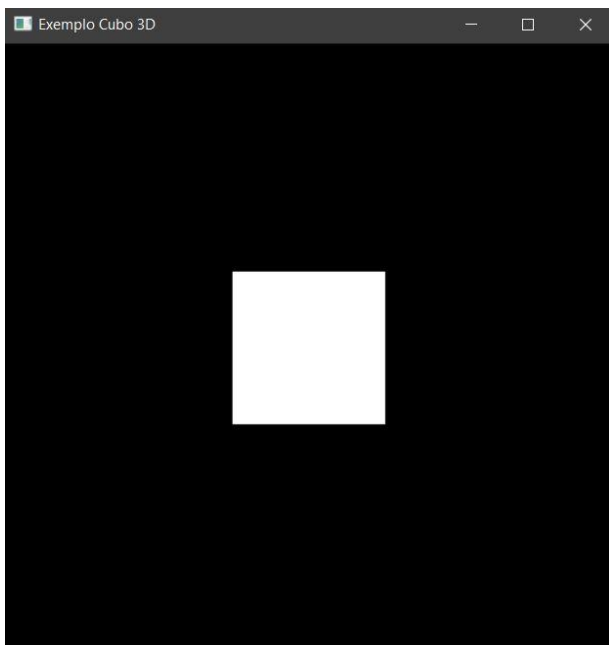
Resultado de outras modificações veja na (Figura 1.2) e na (Figura 1.3):

```
gluLookAt (1.0,0.0,0.0,  
          0.0,0.0,0.0,  
          0.0,1.0,0.0);
```



(Figura 1.2)

```
gluLookAt (-1.0,0.0,0.0,  
          0.0,0.0,0.0,  
          0.0,1.0,0.0);
```



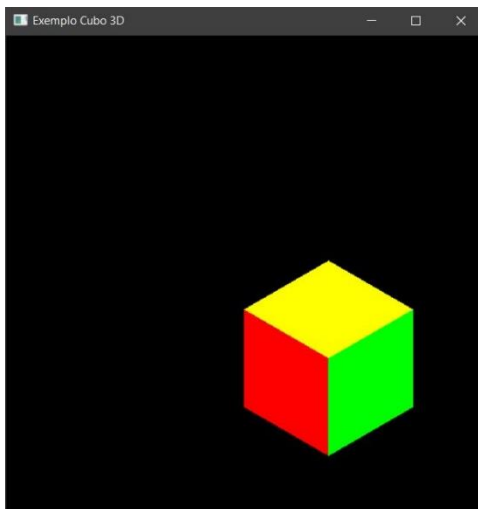
(Figura 1.3)

1.3 Aplicando Translação, Rotação e Escala

A translação é a mudança de um objeto de um lugar para o outro, a rotação é o giro do objeto em torno da origem e a escala é o redimensionamento do objeto, essas três aplicações foram feitas no objeto tridimensional. A primeira modificação a ser feita dentro do programa foi o de translação, para isso foi criado um método transformar,

dentro desse método é feito as três modificações, na parte de translação se passa três parâmetros o (x, y, z), na rotação se passa os parâmetros (ângulo, x, y, z), e na escala os parâmetros são (x, y, z). Veja na (Figura 1.4) apenas o movimento de translação, na (Figura 1.5) apenas o movimento de rotação, na (Figura 1.6) apenas a escala e na (Figura 1.7) são todas as transformações juntas.

```
12  
13 void transformar()  
14 {  
15     glTranslatef(0.70,-0.70,0.0);  
16     glRotatef(45.0,0.0,0.0,1.0);  
17     glScalef(0.50,0.50,0.50);  
18 }  
19
```



(Figura 1.4)

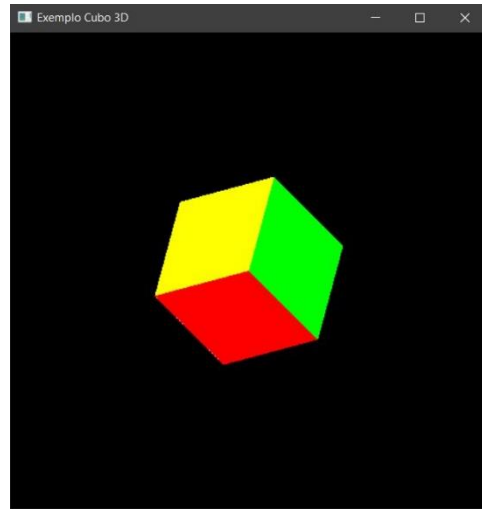


Figura (1.5)

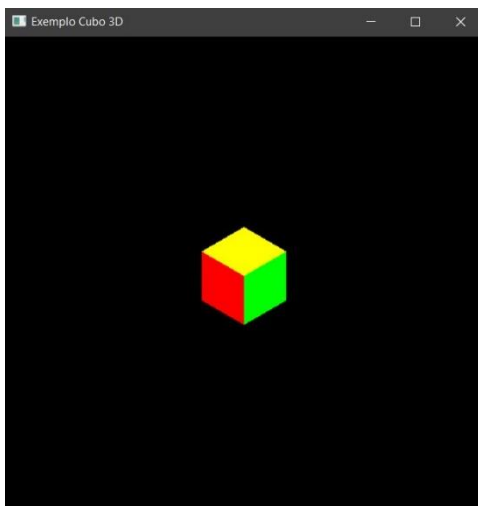


Figura (1.6)

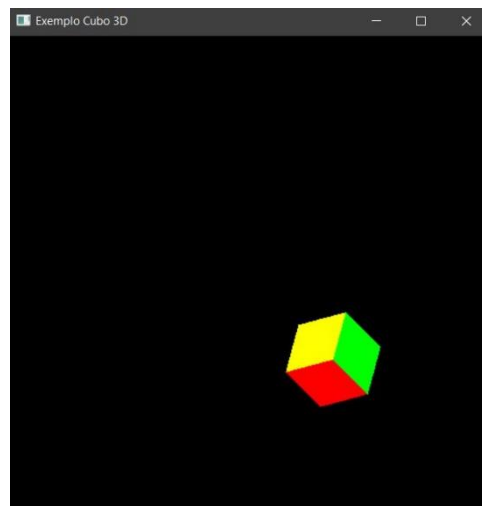


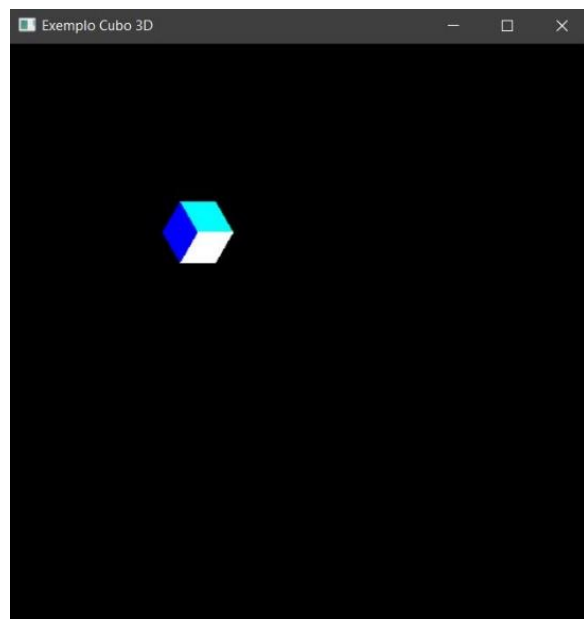
Figura (1.7)

1.4 Outras Modificações

Perceba que aconteceu uma alteração na posição da câmera, foi colocado valores negativos para se conseguir ver o que ta na parte negativa do cubo ou seja (-x, -y, -z), e também foi alterado o método transformar, os parâmetros de translação foi passado um valor negativo pra x e positivo pra y, e no de rotação foi passado um ângulo de -90° ou seja foi rotacionado no sentido horário, e a escala foi diminuída com valores 0.30 no (x, y, z), veja o resultado na (Figura 1.8), outro exemplo pode se ver na (Figura 1.9), com novas mudanças.

```
gluLookAt(-1.0,-1.0,-1.0,
          0.0,0.0,0.0,
          0.0,1.0,0.0);

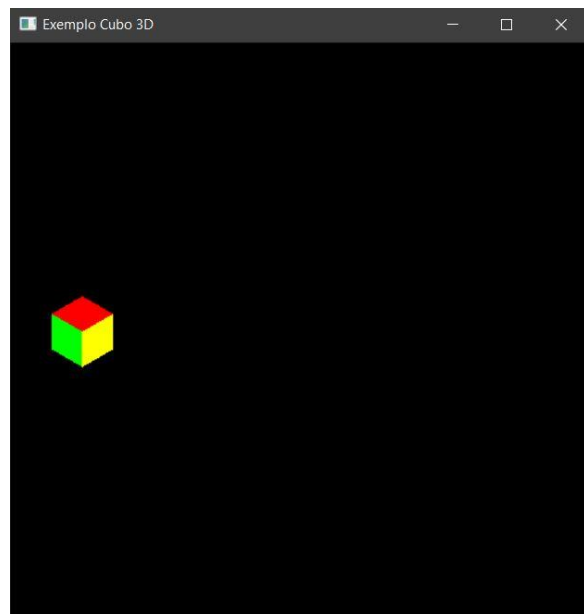
12
13 void transformar()
14 {
15     glTranslatef(-0.70,0.70,0.0);
16     glRotatef(-90.0,0.0,0.0,1.0);
17     glScalef(0.30,0.30,0.30);
18 }
19
```



(Figura 1.8)

```
gluLookAt(1.0,1.0,1.0,
          0.0,0.0,0.0,
          0.0,1.0,0.0);

12
13 void transformar()
14 {
15     glTranslatef(-1.5,0.0,0.0);
16     glRotatef(-120.0,0.0,0.0,1.0);
17     glScalef(0.30,0.30,0.30);
18 }
```



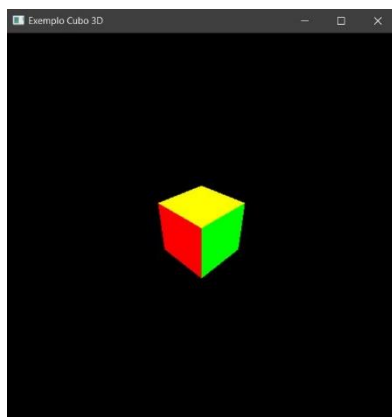
(Figura 1.9)

1.5 Projeção em Perspectiva

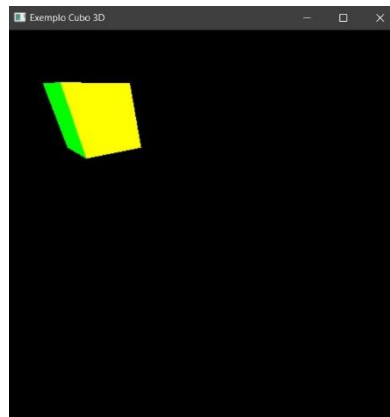
A perspectiva é uma técnica de representação tridimensional que possibilita a ilusão de espessura e profundidade das figuras, dentro do método renderizar foi removido o comando `glOrtho` e colocado o comando `gluPerspective` os parâmetros passados são: (ângulo, proporção da largura e da altura da janela de recorte, plano mais próximo, plano mais distante), a posição da câmera foi alterada para que ela fique um pouco mais distante do objeto, veja na (Figura 2.0) a projeção em perspectiva e nas figuras finais veja os resultados das aplicações de transformações no objeto tridimensional.

```
70
71 void renderizar()
72 {
73     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
74     glMatrixMode(GL_PROJECTION);
75     glLoadIdentity();
76     gluPerspective(90.0, 1.0, 1.0, 4.0);
77     desenhara_cubo();
78     glutSwapBuffers();
79 }
80
```

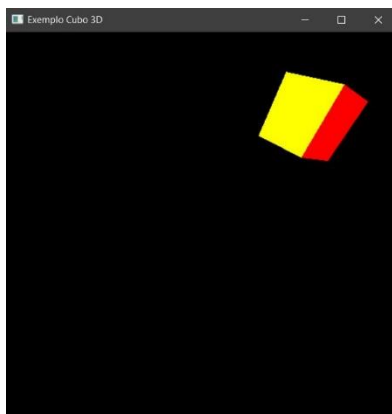
```
gluLookAt(2.0, 2.0, 2.0,
          0.0, 0.0, 0.0,
          0.0, 1.0, 0.0);
```



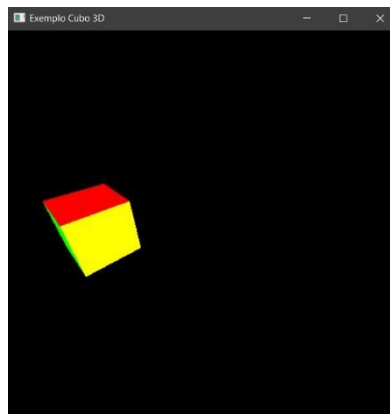
(Figura 2.0)



(Figura 2.1)



(Figura 2.2)



(Figura 2.3)

Link para ver o código no GitHub:

[brunnu-sc/trabalho-objetos-tridimensionais: Computação Gráfica \(github.com\)](https://github.com/brunnu-sc/trabalho-objetos-tridimensionais)

1.5 Referências

Computação Gráfica 05 – Prof. Jorge Cavalcanti (UNIVASF)

Computação Gráfica 07 – Prof. Jorge Cavalcanti (UNIVASF)

<http://www.univasf.edu.br/~jorge.cavalcanti/configcb.html>

Introdução à Computação Gráfica OpenGL Básico (João Paulo & Claudio Esperança)