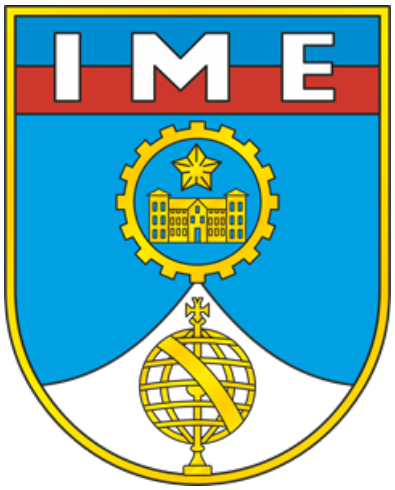
MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
REAL ACADEMIA DE ARTILHARIA, FORTIFICAÇÃO E DESENHO

Seção de Engenharia de Computação / SE 9



Trabalho I

COMPUTAÇÃO GRÁFICA

Professor:
Maj Madeira

Turma: 5º Ano - Computação

1º TEN DANIEL AMBRÓZIO **BRETHERICK** MARQUES
1º TEN **JOSEPH** INÁCIO VIEIRA OLIVEIRA GOMES

Rio de Janeiro, 2024

Conteúdo

1 Introdução

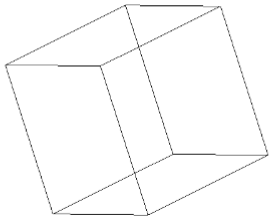
Esse relatório tem por finalidade apresentar os resultados do primeiro trabalho de computação gráfica.

No trabalho, deveriam ser plotadas as figuras requisitadas utilizando-se a ferramenta ”Interpretador Gráfico”, desenvolvida pelos professores da disciplina.

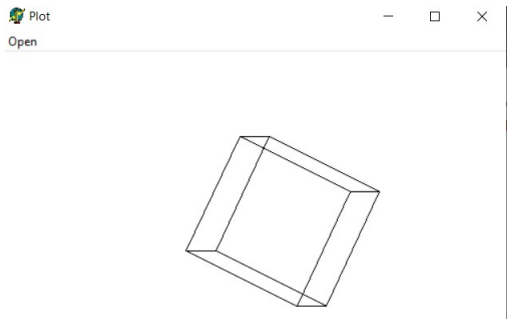
2 Questões

1ª Questão

Utilize o interpretador gráfico para desenhar um cubo, no estilo *wireframe*, rotacionando, sendo apresentado com projeção ortogonal.

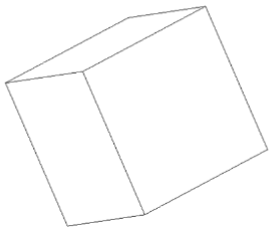


- **Solução:**

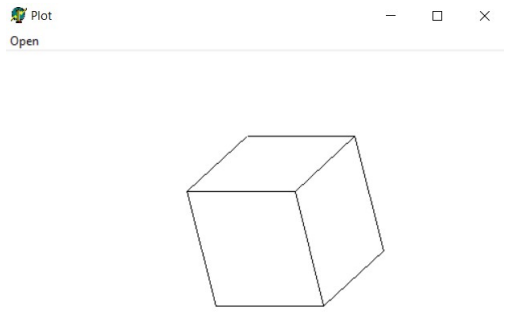


2ª Questão

Repita o exercício da Questão 1, explorando o fato do cubo ser convexo, para eliminar as faces ocultas.



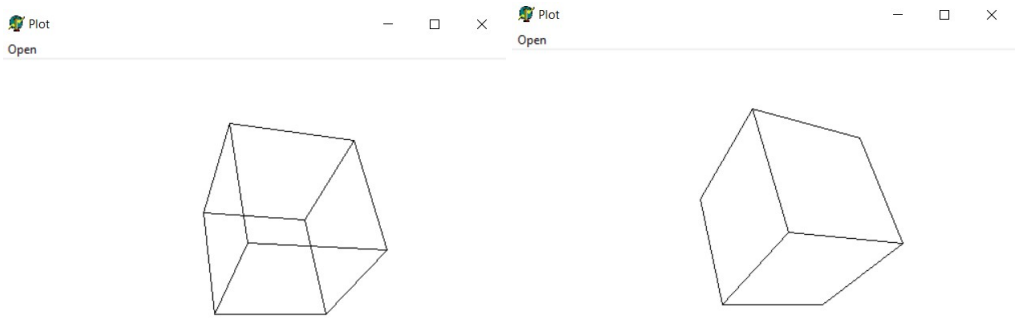
- **Solução:**



3ª Questão

Repita as questões anteriores utilizando projeção perspectiva, no lugar de projeção ortogonal.

- **Solução:**



4ª Questão

Em anexo.

3 Código Fonte

```
import numpy as np

base_z = -2.0
base = 0.0
lado = 0.5
observador = [0.0,0.0,0.0]

#face de baixo
z0 = np.array([base,base,base+base_z])
z1 = np.array([base,base,base+lado+base_z])
z2 = np.array([base,base+lado,base+lado+base_z])
z3 = np.array([base,base+lado,base+base_z])

#face de cima
z4 = np.array([base+lado,base,base+base_z])
z5 = np.array([base+lado,base,base+lado+base_z])
z6 = np.array([base+lado,base+lado,base+lado+base_z])
z7 = np.array([base+lado,base+lado,base+base_z])

vertices =np.array([z0,z1,z2,z3,z4,z5,z6,z7])

rotInit = np.pi/7.0

centro = (z0+z1+z2+z3+z4+z5+z6+z7)/8.0
distancia_focal = 2

XYZ= [0,1,0] #para rotacionar apenas em torno de Y

def createMatrizRot(angle,vetor):
    c = np.cos(angle)
    s = np.sin(angle)
    matriz=np.eye(3)
    Y = np.array([
        [c,0,s],
        [0,1.0,0],
        [-s,0,c]

    ])
    Z = np.array([
        [c,-s,0],
        [s,c,0],
        [0,0,1.0]

    ])
    X= np.array([
        [1.0,0,0],
        [0,c,-s],
        [0,s,c]
    ])
    matrizes = [X,Y,Z]
    for i in range(0,3):
        if vetor[i] > 0.0:
            matriz = matriz@matrizes[i]
    return matriz


```

```
57
58  class WriteToTXT():
59      def __init__(self,txt):
60          self.txt =txt
61
62      def writeLine(self,v1,v2):
63          self.txt.write("line\n")
64          self.txt.write(f"{v1[0]} {v1[1]} {v2[0]} {v2[1]}\n")
65      def writeDelayClear(self,time):
66          self.txt.write(f"delay\n{time}\nclrscr\n")
67
68
69  class Cubo:
70      def __init__(self,vertices,distancia,cuboEmTxt):
71          self.vertices_initial  = vertices
72          self.vertices = vertices
73          self.foco = distancia
74          self.cuboEmTxt= cuboEmTxt
75          self.initial_angle = np.pi/4.0
76          self.final_angle = 4.0*np.pi
77          self.delay = 0.05
78          self.formarArestas()
79          self.formarFaces()
80
81      def formarArestas(self):
82          arestas = []
83
84          for i in range(0,4):
85              arestas.append([(i%4),(i+1)%4])
86          for i in range(0,4):
87              arestas.append([(i%4)+4,(i+1)%4+4])
88          for i in range(0,4):
89              arestas.append([i,i+4])
90          self.arestas = np.array(arestas)
91
92      def formarFaces(self):
93           faces =[]
94           faces.append([self.arestas[0],self.arestas[1],self.arestas[2],self
     .arestas[3]])
95           faces.append([self.arestas[0],self.arestas[8],self.arestas[4],self
     .arestas[9]])
96           faces.append([self.arestas[1],self.arestas[9],self.arestas[5],self
     .arestas[10]])
97           faces.append([self.arestas[2],self.arestas[10],self.arestas[6],
     self.arestas[11]])
98           faces.append([self.arestas[3],self.arestas[11],self.arestas[7],
     self.arestas[8]])
99           faces.append([self.arestas[4],self.arestas[5],self.arestas[6],self
     .arestas[7]])
100          self.faces = np.array(faces)
101
102     def projetarVertice(self,z,f):
103         if f:
104             return np.array([f*z[0]/(z[2]), f*z[1]/(z[2])]);
105         else:
106             return np.array([z[0], z[1]]);
107
108     def projetarTodosVertices(self,projetar= False):
```

```
        vertices = []
        if projetar:
            f = self.foco
        else:
            f = None
        for i in range(0,8):
                vertices.append(self.projetarVertice(self.vertices[i],f))
        return np.array(vertices)

    def rotacionarEmTornoDoProprioCentro(self,angle, transpor = True, vetor
    = centro):
        matrizRot = createMatrizRot(angle,XYZ) #em torno da origem
        centro = self.get_centro()
        if transpor:
            for i in range(0,8):
                vertice = self.vertices[i] - centro
                vertice = vertice @ matrizRot
                self.vertices[i] = vertice + centro
        else:
            for i in range(0,8):
                self.vertices[i] = self.vertices[i] @ matrizRot

    def get_centro_face(self,face):
        centro_face = np.array([0.0,0.0,0.0])
        for i in range(0,4):
            for j in range(0,2):
                centro_face = centro_face+ self.vertices[face[i][j]]
        centro_da_face = np.array(centro_face/8.0)
        return centro_da_face

    def get_centro(self):
        centro = np.array([0.0,0.0,0.0])
        for i in range(0,8):
            centro += self.vertices[i]
        centro = np.array(centro/8.0)
        return centro

    def aparece(self,face,apagar):
        if apagar:

            centroDaFace = self.get_centro_face(face)
            centro = self.get_centro()

            vetor_ortogonal = (centroDaFace - centro)

            produto_escalar = np.dot(vetor_ortogonal,observador-
    centroDaFace)
            if produto_escalar>=0.0:
                return True
            else :

                return False
        else:
            return True


    def escreverCubo(self, projetar,apagar):
        verticesProjetados = self.projetarTodosVertices(projetar)
```

```
        for i in range(0,6):
            face = self.faces[i]

            if self.aparece(face,apagar):
                self.cuboEmTxt.writeLine(verticesProjetados[face[0][0]],
    verticesProjetados[face[0][1]])
                self.cuboEmTxt.writeLine(verticesProjetados[face[1][0]],
    verticesProjetados[face[1][1]])
                self.cuboEmTxt.writeLine(verticesProjetados[face[2][0]],
    verticesProjetados[face[2][1]])
                self.cuboEmTxt.writeLine(verticesProjetados[face[3][0]],
    verticesProjetados[face[3][1]])
        self.cuboEmTxt.writeDelayClear(self.delay)

    def escreverCuboRotacionando(self, projetar=False,apagar=False,
    num_frames=150):
        passo = (self.final_angle - self.initial_angle)/(num_frames*1.0)
        for i in range(0,num_frames):
            self.rotacionarEmTornoDoProprioCentro(passo)
            self.escreverCubo(projetar,apagar)







def restart(base_z=0):

    vertices =np.array([z0,z1,z2,z3,z4,z5,z6,z7])
    for i in range(0,8):
            vertices[i][2]+=base_z
            vertices[i] =  createMatrizRot(rotInit, [0,0,1]) @ vertices[i]
    return vertices

def main():

    vertices= restart()
    with open("./Trabalho_1/cuboProjetado.txt", "w") as arq:
        cuboEmTxt = WriteToTXT(arq)
        cubo = Cubo(vertices,distancia_focal,cuboEmTxt)
        cubo.escreverCuboRotacionando(projetar= True,apagar=False)

    vertices= restart()
    with open("./Trabalho_1/cuboEscondendoProjetado.txt", "w") as arq:
        cuboEmTxt = WriteToTXT(arq)
        cubo = Cubo(vertices,distancia_focal,cuboEmTxt)
        cubo.escreverCuboRotacionando(projetar= True,apagar=True)

    vertices= restart()
    with open("./Trabalho_1/cubo.txt", "w") as arq:
        cuboEmTxt = WriteToTXT(arq)
        cubo = Cubo(vertices,lado,cuboEmTxt)
        cubo.escreverCuboRotacionando()


    vertices= restart(-10.0)
    with open("./Trabalho_1/cuboEscondendo.txt", "w") as arq:
        cuboEmTxt = WriteToTXT(arq)
```

```
218         cubo = Cubo(vertices,lado,cuboEmTxt)
219         cubo.escreverCuboRotacionando(projetar= False,apagar=True)
220 
221 
222 if __name__ == "__main__":
223     main()
```