Universidade Federal de Ouro Preto - UFOP
Instituto de Ciências Exatas e Biológicas - ICEB
Departamento de Computação - DECOM
Ciência da Computação

Atividade 3 - Transformações geométricas 2D e 3D

BCC327 - Computação Gráfica

Matheus Peixoto Ribeiro Vieira - 22.1.4104
Professor: Rafael Alves Bonfim de Queiroz

Ouro Preto
30 de novembro de 2024

Sumário

Lista de Figuras

Lista de Códigos Fonte

1 Introdução

Para este trabalho, são solicitadas aplicações gráficas que contemplem os temas de transformação 2D e 3D de translação, escala e/ou rotação.

1.1 Especificações da máquina

A máquina onde o desenvolvimento foi realizado possui a seguinte configuração:

- Processador: Intel Core i5-9300H
- Memória RAM: 16Gb.
- Sistema Operacional: WSL 2.0 com Ubuntu 22.04.5 LTS

2 Transformação em 2D

A primeira atividade solicita a implementação de uma aplicação que realiza uma transformação geométrica em um objeto 2D. Para isso, a transformação escolhida foi a translação.

Diferente de outras atividades, o motor gráfico escolhido foi o PyGame [4], pois dado o uso do Python e com uma linguagem mais simples de ser entendida, a aplicação e fixação dos conceitos de computação gráfica tornam-se mais proveitosos.

Na atividade, foi criada uma tela em branco com tamanho de 800x600 e carregada a imagem do personagem Mario que pode ser vista na Figura 1

Figura 1: Imagem do Mario utilizada
Fonte: [9]

Assim como no OpenGL, o PyGame também possui um loop para capturar eventos e desenhar na tela os elementos. Durante essa repetição, o usuário pode apertar teclas do teclado, que podem ser obtidas a partir do `pygame.key.get_pressed()` [5].

Dessa forma, foram criadas duas variáveis com valor zero, uma ‘x‘ e outra ‘y‘. Então, quando o usuário pressona a tecla ‘w‘, o valor de ‘y‘ incrementa em um, quando aperta ‘s‘, o valor de ‘y‘ decrementa em uma unidade, permitindo uma movimentação no eixo Y. Agora, quando aperta a tecla ‘a‘, o valor de ‘x‘ decrementa em uma unidade e, quando aperta a tecla ‘d‘ ela incrementa em uma unidade, permitindo a movimentação no eixo X.

Quando o valor de X ou de Y é modificado, chamamos a função de translação para calcular a nova posição do Mario, tendo como ponto de partida a posição central do mesmo na tela, sendo que este será modificado após a finalização dos cálculos.

O código 1 realiza a translação do objeto, recebendo uma tupla para as antigas coordenadas X e Y e o deslocamento a ser realizado. Por fim, é verificado se os novos valores não ultrapassarão a tela, pois, se forem, a movimentação será limitada.

```
1 def translacao(antigo, novo):
2     x, y = antigo
3     dx, dy = novo
```

```
    matriz_translacao = np.array([
        [1., 0., dx],
        [0., 1., dy],
        [0., 0., 1.],
    ])
    posicao = np.array([x, y, 1]).T
    nova_posicao = np.dot(matriz_translacao, posicao)[:2]

    # Impedindo que o X e o Y ultrapassem os limites da tela
    novo_x = nova_posicao[0]
    if novo_x > largura: novo_x = largura
    elif novo_x < 0: novo_x = 0

    novo_y = nova_posicao[1]
    if novo_y > altura: novo_y = altura
    elif novo_y < 0: novo_y = 0

    return novo_x, novo_y
```

Código 1: Função de translação

A figura 2 o Mario após uma sequência de movimentações, tendo começado no ponto central da tela. Ademais, são mostradas diferentes matrizes de translação modificadas, a posição original do Mario e o resultado da multiplicação das matrizes que geram a transformação de translação.

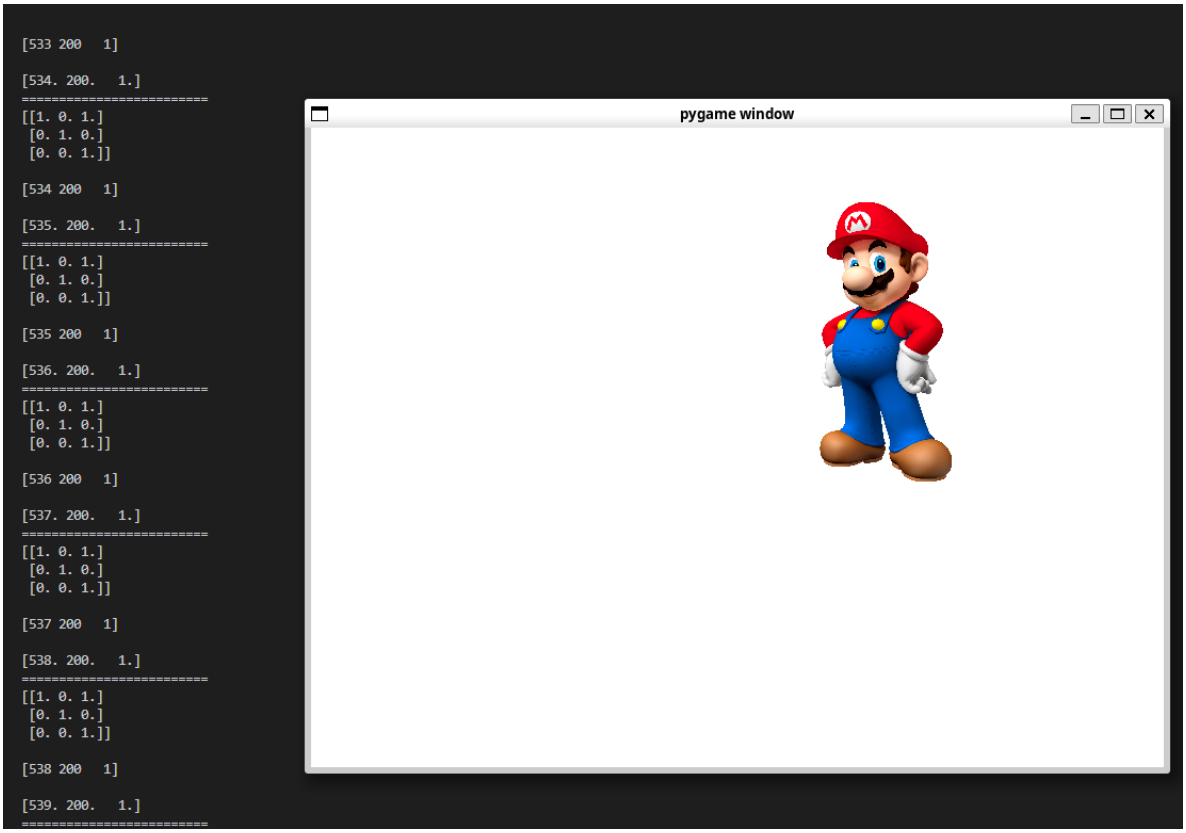

Figura 2: Mario transladado

3 Transformação em 3D

O PyGame não permite a programação nativa de objetos 3D [1]. Então, para realizar a atividade com objetos em 3D será utilizado a biblioteca PyOpenGL, que conecta o python com a API do OpenGL

[3]. Sendo que, para um contato inicial com ele foi seguido o tutorial de [2]

Para essa tarefa, diferente do tutorial que criou um cubo, aqui foi criado uma piramide onde cada vértice e aresta foram manualmente determinados, como pode ser visto no código 2.

```
vertices = [
    (0, 1, 0),  # Topo
    (-1, -1, 1), # Frente esquerda
    (1, -1, 1), # Frente direita
    (-1, -1, -1), # tras esquerda
    (1, -1, -1) # tras direita
]
# Definicao das arestas para conectar a base e o topo
arestas = (
    (0, 1), (0, 2), (0, 3), (0, 4),
    (1, 2), (1, 3), (2, 4), (3, 4)
)
```

Código 2: Pirâmide

Em seguida, é criada uma função para a geração da pirâmide informando que será utilizado o OpenGL e passando os vértices e arestas e informando, com o `GL_LINES` que estão sendo criadas linhas entre as coordenadas informadas, como pode ser visto no código 3

```
def Pyramid():
    glBegin(GL_LINES)
    for edge in arestas:
        for vertex in edge:
            glVertex3fv(vertices[vertex])
    glEnd()
```

Código 3: Geração pirâmide

Para ajustar a visualização do objeto na tela definindo a perspectiva do mesmo, seu aspect-ratio e os limites dos valores para sua visualização, definimos com o comando `gluPerspective(45, (display[0]/display[1]), 0.` e em seguida executamos o `glTranslatef(0.0, 0.0, -5)` para ajustar a posição do objeto, pois caso contrário ele estaria muito "próximo"da tela.

Assim, como na seção anterior onde o objeto foi movido ao redor do eixo X e Y pelas teclas 'w', 'a', 's' e 'd', agora também iremos girar o objeto ao redor do seu próprio eixo Y, indo em uma direção com a seta do teclado para a esquerda e indo para a direção oposta com a seta para direita.

Possuindo uma alteração no valor de movimentação de x ou y, a função de translação será chamada. Aqui também não foi utilizada uma função pré-programada das bibliotecas para tal transformação. Dessa forma, a nova posição de cada vértice é calculada e modificada pelo código 4

```
def translacao(dx, dy, dz):
    matriz_translacao = np.array([
        [1., 0., 0., dx],
        [0., 1., 0., dy],
        [0., 0., 1., dz],
        [0., 0., 0., 1.],
    ])
    for i, (x, y, z) in enumerate(vertices):
        vertice = np.array([x, y, z, 1])
        vertices[i] = np.dot(matriz_translacao, vertice.T)[:3]
```

Código 4: Translação 3D

De forma semelhante, o mesmo ocorre para a rotação Y, como pode ser visto no código 5

```
def rotacaoY(graus):
    radianos = math.radians(graus)
    cos = math.cos(radianos)
    sen = math.sin(radianos)
    matriz_rotacao = np.array([
        [cos, 0, -sen, 0],
        [  0, 1,    0, 0],
```

```
        [sen, 0,  cos, 0],
        [0,   0,    0, 1],
    ])
    for i, (x, y, z) in enumerate(vertices):
        vertice = np.array([x, y, z, 1])
        vertices[i] = np.dot(matriz_rotacao, vertice.T)[:3]
```

Código 5: Rotação eixo Y

A figura 3 mostra a pirâmide renderizada inicialmente enquanto a 4 mostra a pirâmide após ser transladada e rotacionada ao redor do seu eixo Y.

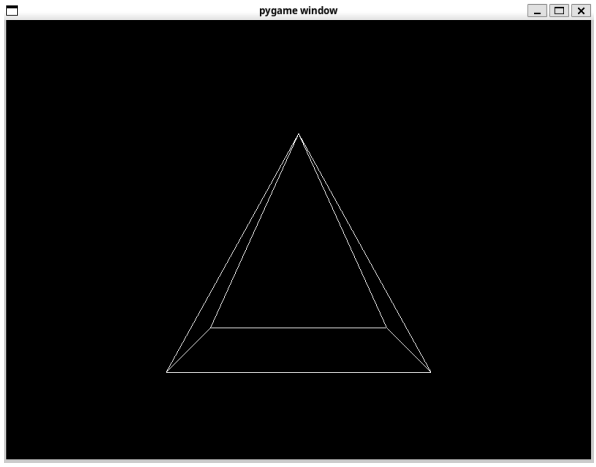

Figura 3: Pirâmide Inicial

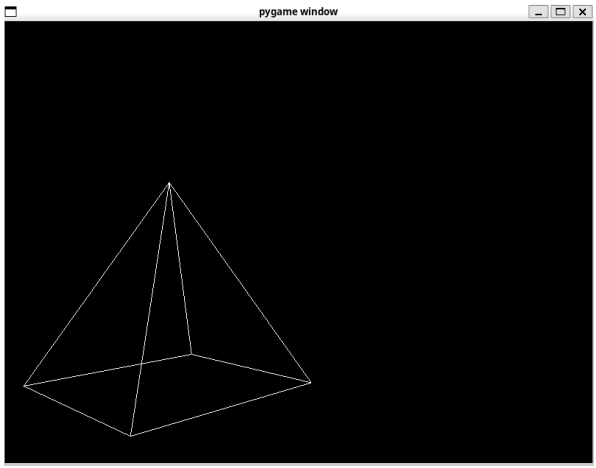

Figura 4: Pirâmide Final

4 Composição de transformação 2D

Para o 2D, o Mario será utilizado novamente e, agora, os eventos de teclado serão modificados. Sempre que a tecla 'w' for pressionada, iremos girar o Mario em 90 graus e aumentar o seu tamanho em 10%. Agora, caso seja pressionada o 's', ele será girado no sentido contrário, mas pela mesma angulação e o seu tamanho será diminuído em 10%.

Para realizar a composição, a fim de explorar mais a biblioteca PyGame, serão utilizadas as funções `pygame.transform.rotate` [6] seguido da `pygame.transform.scale_by` [8], como pode ser observado no código 6

```
def composicao(angulo, escala, surface):
    composta =  pygame.transform.scale_by(
        pygame.transform.rotate(surface, angulo),
        escala
    )
    return composta
```

Código 6: Rotacao e escala

Vale ressaltar que a biblioteca já possui uma função pronta que realiza tal operação, sendo esta a `pygame.transform.rotozoom()` [7], mas que, por motivos didáticos, não foi utilizada.

Dessa forma, a figura 5 mostra como ficou o Mario após alguns pressionamentos da tecla 'w' que ocasionou no aumento de seu tamanho e sua rotação em sentido anti-horário.

Figura 5: Mario rotacionado e escalado

5 Composição de transformação 3D

Para a composição em 3D, a pirâmide foi mantida, mas modificações foram feitas no código. Agora não haverá mais o movimento de translação, somente o de rotação. Todavia, a rotação será composta pela rotação dos três eixos, gerando a seguinte função:

$$C = R_x(\theta)R_y(\theta)R_z(\theta)$$

Sendo que ela foi implementada pelo código 7 onde as transformações foram manualmente implementadas.

```
def composicao(graus):
    radianos = math.radians(graus)
    cos = math.cos(radianos)
    sen = math.sin(radianos)
    matriz_rotacaoX = np.array([
        [1,   0,    0, 0],
        [0, cos, -sen, 0],
        [0, sen,  cos, 0],
        [0,   0,    0, 1]
    ])
    matriz_rotacaoY = np.array([
        [cos, 0, -sen, 0],
        [  0, 1,    0, 0],
        [sen, 0,  cos, 0],
        [0,   0,    0, 1],
    ])
    matriz_rotacaoZ = np.array([
```

```
        [cos, -sen, 0, 0],
        [sen,  cos, 0, 0],
        [  0,    0, 1, 0],
        [0,      0, 0, 1],
    ])

    # f(x) = rotacaoX * rotacaoY * rotacaoZ
    for i, (x, y, z) in enumerate(vertices):
        vertice = np.array([x, y, z, 1])
        rz = np.dot(matriz_rotacaoZ, vertice)
        ry = np.dot(matriz_rotacaoY, rz)
        rx = np.dot(matriz_rotacaoX, ry)
        vertices[i] = rx[:3]
```

Código 7: Rotações compostas

Dessa forma, após um tempo pressionando uma das setas do teclado, a pirâmide foi rotacionada e o resultado pode ser observado na figura 6

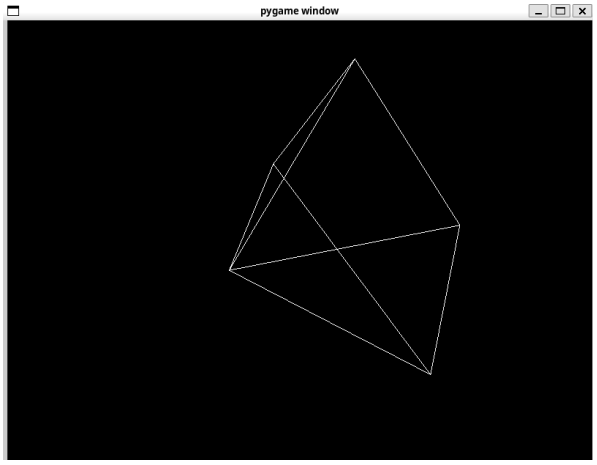

Figura 6: Pirâmide rotacionada

6 Conclusão

Com o fim da atividade pode-se concluir o seu objetivo de fixar os conteúdos sobre transformações geométricas, principalmente com a escolha de implementar manualmente as transformações e composições utilizadas, exceto no caso da composição 2D onde foi escolhido explorar mais a biblioteca do PyGame.

Referências

[1] Stack Overflow. Does pygame do 3d? https://stackoverflow.com/questions/4865636/does-pygame-do-3d. [Online; acessado em 29-Novembro-2024].

[2] Stack Overflow. Opengl with pyopengl introduction and creation of rotating cube. https://pythonprogramming.net/opengl-rotating-cube-example-pyopengl-tutorial/. [Online; acessado em 29-Novembro-2024].

[3] Stack Overflow. Pyopengl - the python opengl binding. https://pyopengl.sourceforge.net/. [Online; acessado em 29-Novembro-2024].

[4] PyGame. Pygame intro. https://www.pygame.org/docs/tut/PygameIntro.html. [Online; acessado em 29-Novembro-2024].

[5] PyGame. Pygame key. https://www.pygame.org/docs/ref/key.html#pygame.key.get_pressed. [Online; acessado em 29-Novembro-2024].

[6] PyGame. rotate. https://www.pygame.org/docs/ref/transform.html#pygame.transform.rotate. [Online; acessado em 29-Novembro-2024].

[7] PyGame. rotozoom. https://www.pygame.org/docs/ref/transform.html#pygame.transform.rotozoom. [Online; acessado em 29-Novembro-2024].

[8] PyGame. scale_by. https://www.pygame.org/docs/ref/transform.html#pygame.transform.scale_by. [Online; acessado em 29-Novembro-2024].

[9] wcwjunkbox. Super mario png 2022. https://www.deviantart.com/wcwjunkbox/art/Super-Mario-PNG-2022-941434461. [Online; acessado em 29-Novembro-2024].