Computação Gráfica

Trabalho 3

Aluno: João Vitor Branquinho Ribeiro
Matrícula: 216.031.144

1- Meu objeto do trabalho 1 é um Prisma Pentagonal.

Primeiramente, defino de onde está vindo a luz. Como o enunciado pede maior incidência na face a frente, defino que a luz está vindo diretamente paralela ao eixo Z positivo:

```python
# ======== CONFIGURAÇÃO DA LUZ ========
# luz vindo da frente (direção Z positiva)
direcao_luz = np.array([0, 0, 1])
direcao_luz = direcao_luz / np.linalg.norm(direcao_luz) # normaliza
```

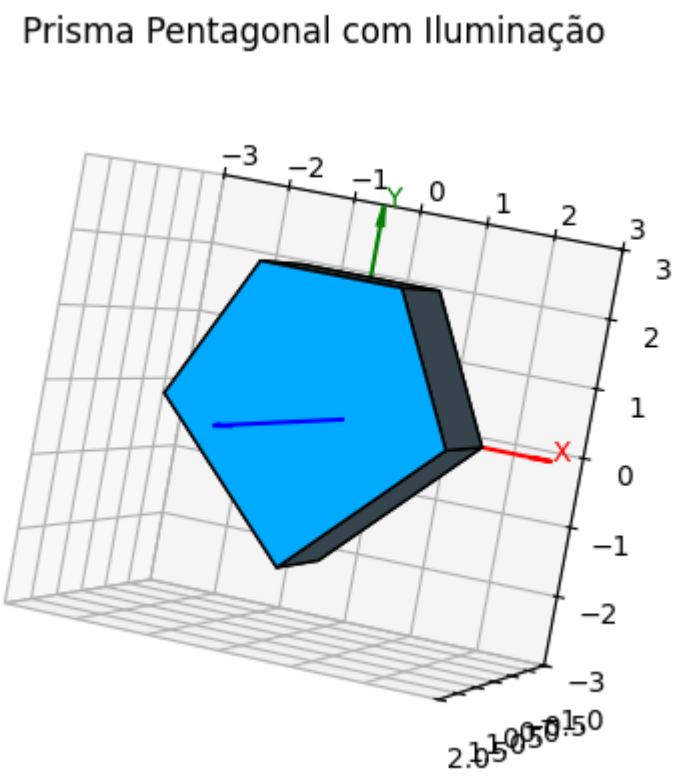
Como precisarei calcular a normal de cada face, criei um método para isso.

```python
# ======== CÁLCULO DA NORMAL DE UMA FACE ========
Windsurf: Refactor | Explain | ×
def face_normal(face):
    """
    Calcula a normal de uma face.
    """

    # define os vértices da face
    v0 = np.array(vertices[face[0]])
    v1 = np.array(vertices[face[1]])
    v2 = np.array(vertices[face[2]])

    # calcula os vetores de aresta
    vetor1 = v1 - v0
    vetor2 = v2 - v0

    # calcula a normal
    normal = np.cross(vetor1, vetor2) # produto vetorial
    normal = normal / np.linalg.norm(normal) # normaliza
    return normal
```

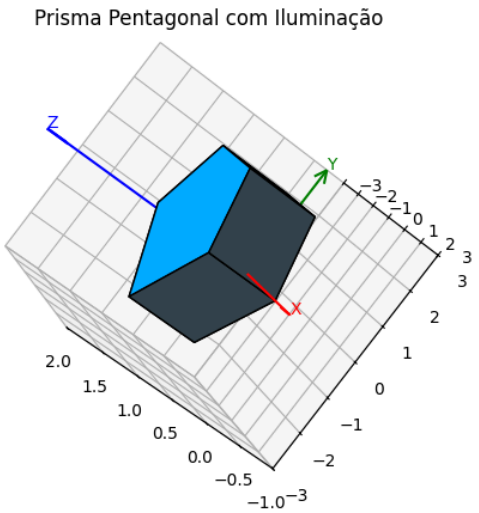
Finalmente, faço o cálculo da cor, levando em consideração o cosseno do ângulo entre a normal e a luz, com o hue definido em um tom de azul, e com a saturação e intensidade variando:

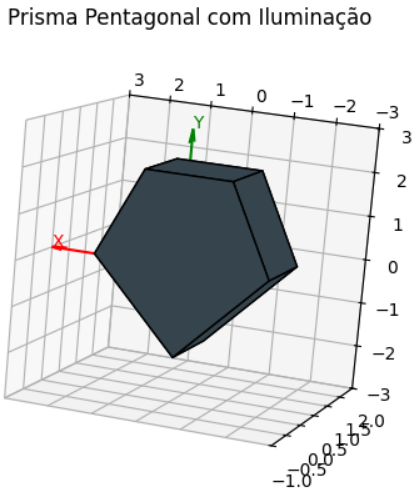
```python
# ======== CÁLCULO DA ILUMINAÇÃO (Luz ambiente + direta) ========
Windsurf: Refactor | Explain | X
def compute_color(normal_face):
    """
    Calcula a cor de uma face.
    Recebe a normal da face como parâmetro
    """
    hue = 200 / 360  # tom azul
    luz_ambiente = 0.3 # luz ambiente de 30%
    cos_theta = np.dot(normal_face, direcao_luz) # cosseno do angulo entre a normal e a luz
    luz_direta = max(cos_theta, 0)

    intensidade = luz_ambiente + 0.8 * luz_direta
    intensidade = min(intensidade, 1)  # garantir limite máximo de 100%

    saturacao = luz_ambiente + 0.8 * luz_direta
    saturacao = min(saturacao, 1)  # garantir limite máximo de 100%

    # Hue constante, saturação e intensidade variam com a iluminação
    r, g, b = colorsys.hsv_to_rgb(hue, saturacao, intensidade)
    return (r, g, b)
```

Assim ficou o objeto com iluminação vindo paralela ao eixo Z (positivo):

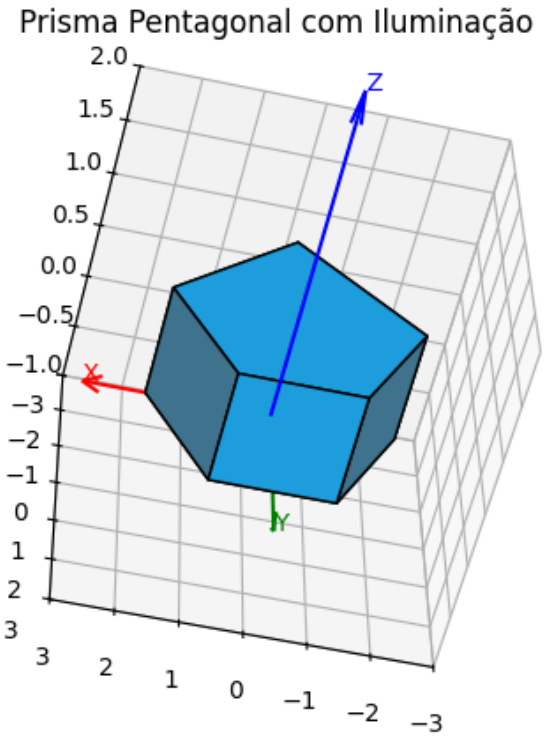


Repare que todas as outras faces apenas recebem iluminação ambiente de 30%.



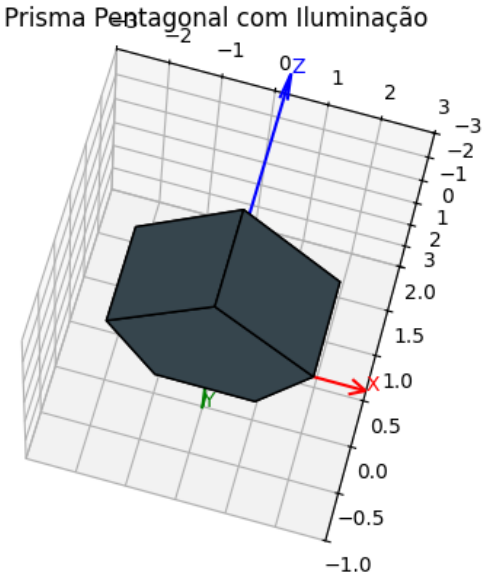


Para fins de teste do algoritmo, faço a iluminação também existir, tanto do eixo Z positivo quanto também para o eixo Y positivo:



Repare que tanto a face do Z positivo quanto a do Y positivo ficaram com 100% de iluminação. Além disso, as faces adjacentes ficaram com iluminação parcial, por fazer um ângulo entre a normal da face e a direção da luz.

Já as faces que não ficaram iluminadas, recebem apenas luz ambiente (30%).



4- Como estou fazendo o trabalho sozinho, não possuo objeto de um colega para plotar junto ao meu.

Mas, para aplicar o algoritmo do pintor, implementei o seguinte método:

```python
# ======== CÁLCULO DA PROFUNDIDADE PARA O ALGORITMO DO PINTOR ========
Windsurf: Refactor | Explain | X
def profundidade_face(face):
    """
    Calcula uma média de profundidade para cada face.
    Prioriza o eixo Z (profundidade), que representa a distância do observador.
    """
    zs = [vertices[i][2] for i in face]
    ys = [vertices[i][1] for i in face]
    xs = [vertices[i][0] for i in face]
    return np.mean(zs) + np.mean(ys)
```

Esse método calcula a média da profundidade de cada face, conseguindo assim garantir que as primeiras serão as mais distantes do observador.

```python
# cria uma lista ordenada de índices das faces,
# ordenada da face mais distante para a mais próxima (algoritmo do pintor)
face_order = sorted(
    range(len(faces)),
    key=lambda idx: profundidade_face(faces[idx]),
    reverse=True  # desenha as mais distantes primeiro
)
```