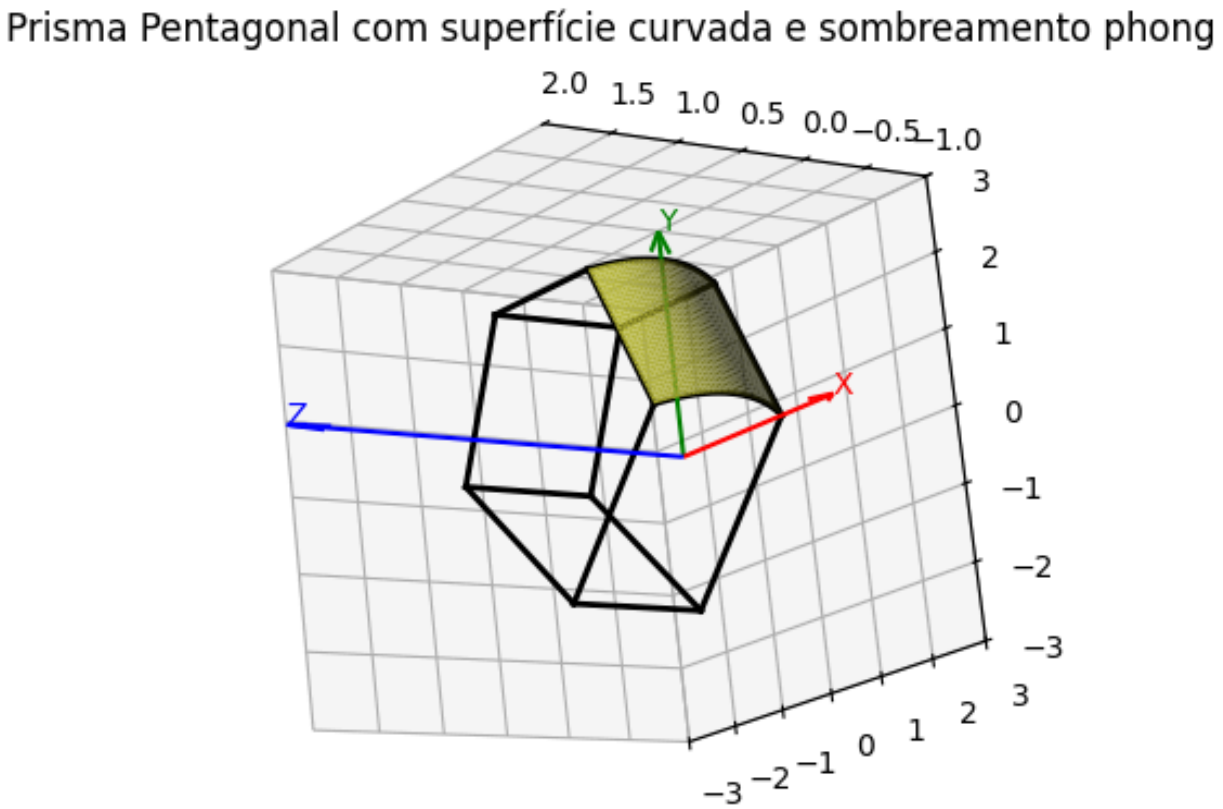
Computação Gráfica

Trabalho 4

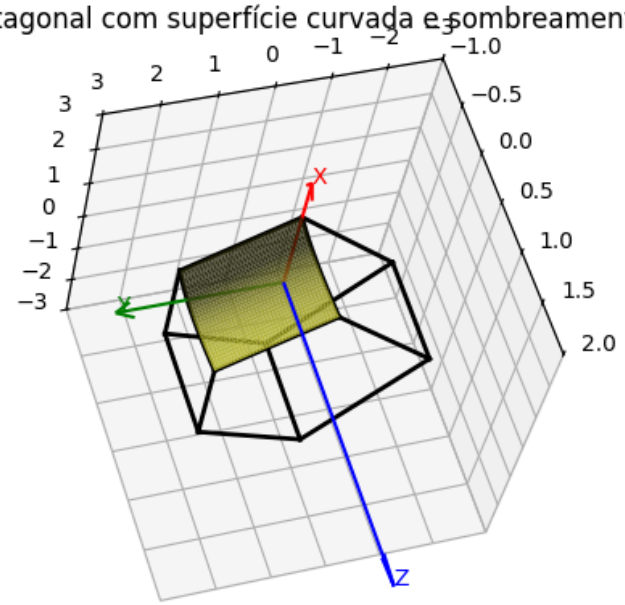
Aluno: João Vitor Branquinho Ribeiro
Matrícula: 216.031.144

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

Para obter o resultado esperado no enunciado, foi substituída uma face do prisma por uma superfície curvada parabólica, definida a partir de interpolações paramétricas.



Prisma Pentagonal com superfície curvada e sombreamento phong

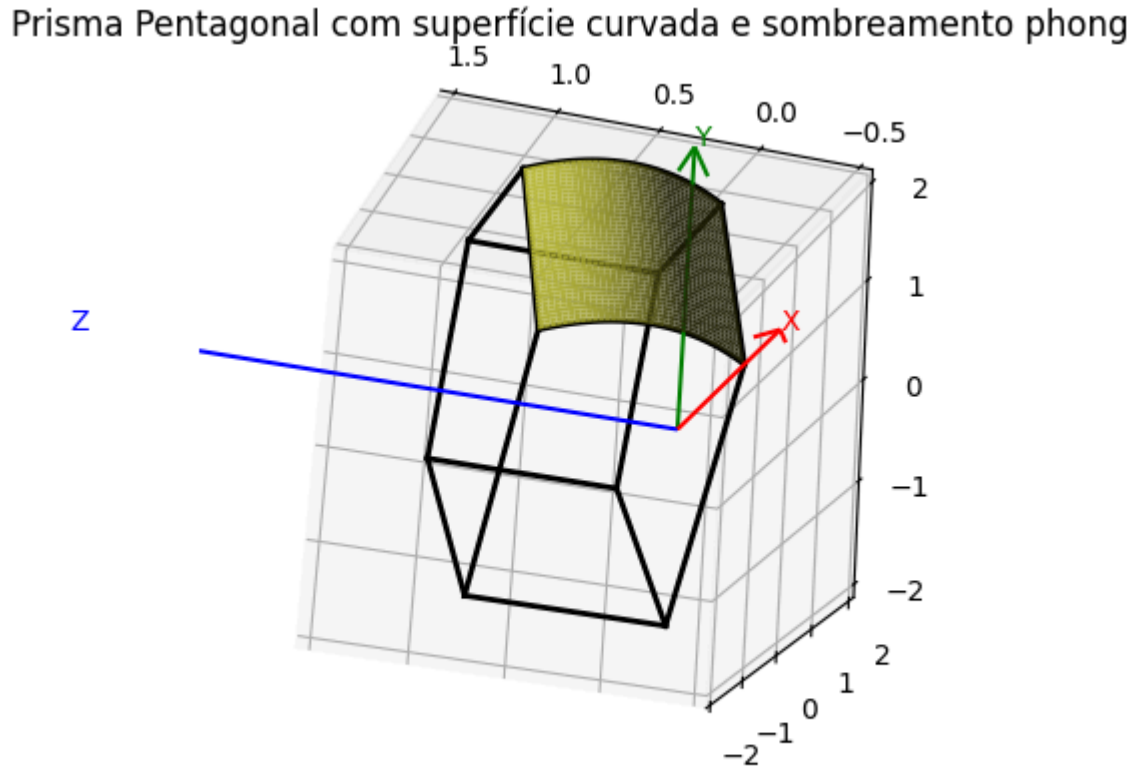



```
# desenha arestas (exceto as da face curva)
for (i, j) in arestas:
    if (i, j) in arestas_remover or (j, i) in arestas_remover:
        continue
    x = [vertices[i][0], vertices[j][0]]
    y = [vertices[i][1], vertices[j][1]]
    z = [vertices[i][2], vertices[j][2]]
    ax.plot(x, y, z, 'k', linewidth=2)

# vértices da face curva
v0 = np.array(vertices[0])
v1 = np.array(vertices[1])
v6 = np.array(vertices[6])
v5 = np.array(vertices[5])
```

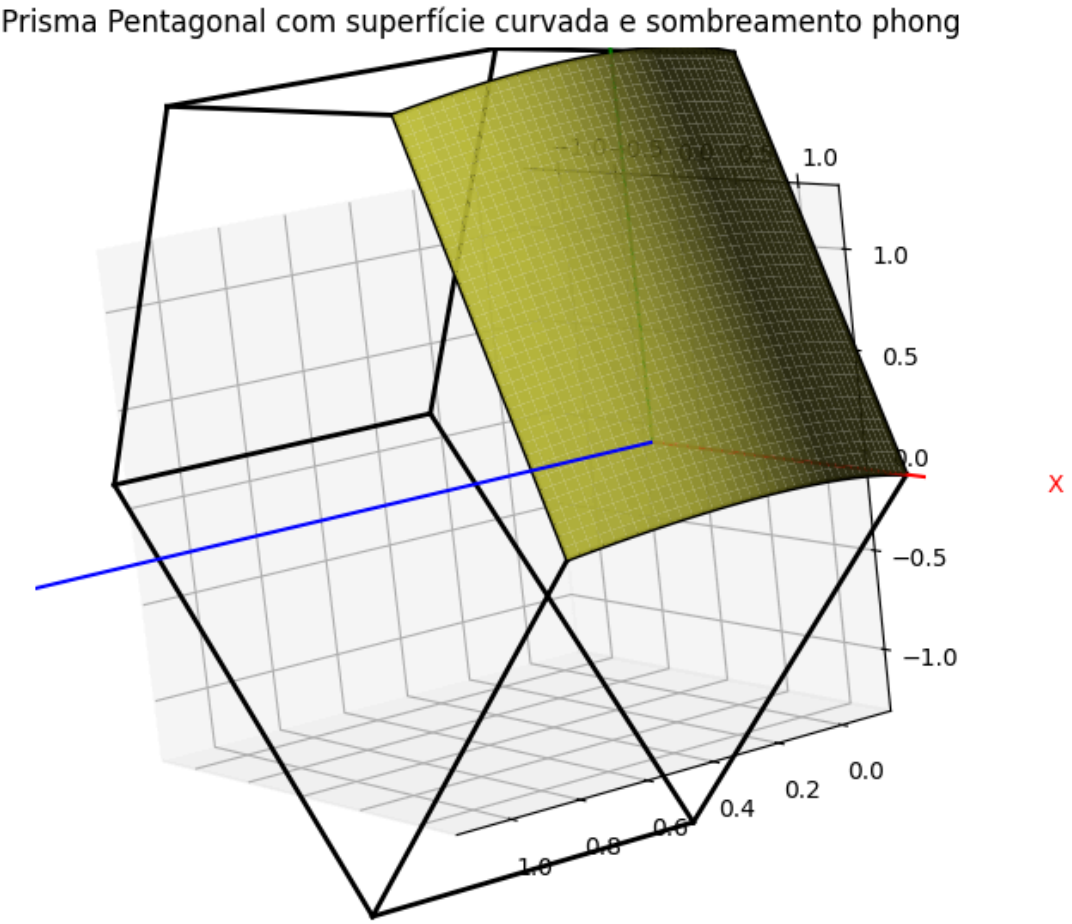
```
# desenha a superfície curvada
for i in range(n):
    linha = np.linspace(borda_baixo[i], borda_cima[i], n)
    t = np.linspace(0, 1, n)
    curva = curvatura * (t - 0.5)**2 # curva parabólica
    normal = np.cross(v1-v0, v5-v0)
    normal = normal / np.linalg.norm(normal)
    deslocamento = curvatura * 0.25 # ajuste do deslocamento da curva para encaixar nas arestas da face
    linha_curva = linha - curva[:, None] * normal + deslocamento * normal
    X.append(linha_curva[:, 0])
    Y.append(linha_curva[:, 1])
    Z.append(linha_curva[:, 2])

X = np.array(X)
Y = np.array(Y)
Z = np.array(Z)
```

2- Para aparência do polimento foi usado brilho especular, ou seja, reflexos que variam com o ângulo de visão e iluminação. (página 229 do livro). Para isso, foi implementado com base no sombreamento de phong completo, utilizando os 3 componentes de ambiente, difuso e especular (cada um com seu ka, kd e ks respectivo).

Repare como fica o objeto ao definir uma iluminação vinda do eixo Z:



Se dermos zoom, podemos notar que, para cada ponto da superfície, rodamos um método `sombreamento_phong_superficie()`
que calcula a normal da superfície naquele ponto. A normal então é usada para calcular ka, kd e ks, e finalmente a cor final daquele ponto com sua iluminação correta.



```python
def sombreamento_phong_superficie(X, Y, Z, pos_luz, pos_observador, cor_luz, ka, kd, ks, brilho):
    """
    Aplica o modelo de sombreamento de Phong a uma superfície 3D definida pelas coordenadas X, Y e Z.

    Create Jira Issue
    Este método calcula a cor de cada ponto da superfície utilizando os componentes ambiente, difuso e especular
    do modelo de iluminação de Phong. A iluminação é feita considerando uma fonte de luz pontual e uma superfície polida.

    Parâmetros:
    ----------
    X, Y, Z : np.ndarray
        Matrizes 2D representando as coordenadas dos pontos da superfície em 3D.
    pos_luz : np.ndarray
        Vetor (x, y, z) com a posição da fonte de luz.
    pos_observador : np.ndarray
        Vetor (x, y, z) com a posição do observador (câmera).
    cor_luz : np.ndarray
        Cor da luz como vetor RGB com valores entre 0 e 1. (Ex: [1.0, 1.0, 0.3] para luz amarela)
    ka : float
        Coeficiente de refletância ambiente da superfície.
    kd : float
        Coeficiente de refletância difusa da superfície.
    ks : float
        Coeficiente de refletância especular da superfície.
    brilho : float
        Expoente de brilho (shininess) usado na reflexão especular. Quanto maior, mais focado é o brilho.

    Retorno:
    -------
    np.ndarray
        Matriz 3D (n, m, 3) contendo os valores de cor RGB calculados para cada ponto da superfície.
    """
    n, m = X.shape
    rgb = np.zeros((n, m, 3))

    Windsurf: Refactor | Explain | Generate Docstring | X
    def calcula_normal(i, j):
```

```python
    def calcula_normal(i, j):
        if i < n-1:
            dXdi = np.array([X[i+1,j]-X[i,j], Y[i+1,j]-Y[i,j], Z[i+1,j]-Z[i,j]])
        else:
            dXdi = np.array([X[i,j]-X[i-1,j], Y[i,j]-Y[i-1,j], Z[i,j]-Z[i-1,j]])
        if j < m-1:
            dXdj = np.array([X[i,j+1]-X[i,j], Y[i,j+1]-Y[i,j], Z[i,j+1]-Z[i,j]])
        else:
            dXdj = np.array([X[i,j]-X[i,j-1], Y[i,j]-Y[i,j-1], Z[i,j]-Z[i,j-1]])
        normal = np.cross(dXdi, dXdj)
        normal = normal / (np.linalg.norm(normal) + 1e-8)
        return normal

    for i in range(n):
        for j in range(m):
            posicao = np.array([X[i, j], Y[i, j], Z[i, j]])
            normal = calcula_normal(i, j)
            para_luz = pos_luz - posicao
            para_luz = para_luz / np.linalg.norm(para_luz)
            para_observador = pos_observador - posicao
            para_observador = para_observador / np.linalg.norm(para_observador)
            ambiente = ka * cor_luz
            difusa = kd * cor_luz * max(np.dot(normal, para_luz), 0)
            reflexao = 2 * np.dot(normal, para_luz) * normal - para_luz
            angulo_especular = max(np.dot(reflexao, para_observador), 0)
            especular = ks * cor_luz * (angulo_especular ** brilho)
            cor = ambiente + difusa + especular
            rgb[i, j, :] = np.clip(cor, 0, 1)
    return rgb
```

Referências:

_Livro texto (em especial cap 5.4.3, 5.4.4.3 e 3.2)

_https://matplotlib.org/stable/gallery/mplot3d/
(documentação da lib matplotlib)

Código do trabalho:

https://github.com/joaovitorbranq/Trabalho-CG