Visualização de Informação

Universidade de Aveiro

João Ferreira, Rui Campos

Visualização de Informação

Lesson 2

Universidade de Aveiro

João Ferreira, Rui Campos
(103625) joaop.ferreira@ua.pt, (103709) ruigabriel2@ua.pt

Dezembro 2024

Índice

Chapter 1

Introduction

Este é o relatório desenvolvido para a cadeira de Visualização de Informação (VI), onde exploramos as propriedades e funcionalidades do VTK (Visualization Toolkit). Ao longo do relatório implementamos diversos exemplos práticos para compreender os conceitos de manipulação de objetos 3D.

O objetivo principal foi adquirir uma compreensão aprofundada da visualização do VTK, utilizando diferentes primitivas geométricas e propriedades de renderização para criar cenas interativas. Além disso, investigamos como alterações específicas, como mudanças de cores, transparência impactam a visualização final.

Ao longo do relatório, documentamos o código desenvolvido, os experimentos realizados e as observações das representações gráficas geradas.

Chapter 2

Multiple actors

Foram criados dois cones, cada um com propriedades distintas, permitindo observar como as propriedades compartilhadas e as transformações afetam a sua renderização.

2.1 Modificação do Primeiro Cone

O primeiro cone teve suas propriedades ajustadas diretamente no ator utilizando os métodos da classe vtkActor

```
coneActor = vtkActor()
coneActor.SetMapper(coneMapper)
coneActor.GetProperty().SetColor(0.2, 0.63, 0.79)   # Cor azul clara
coneActor.GetProperty().SetDiffuse(0.7)
coneActor.GetProperty().SetSpecular(0.4)
coneActor.GetProperty().SetSpecularPower(20)
```

Fig.1: Criação do primeiro cone

2.2 Modificação do Segundo Cone

O segundo cone foi configurado utilizando um objeto do tipo vtkProperty.

```
# ======== Segundo Cone ======== #
# Propriedades compartilhadas para o segundo cone
property = vtkProperty()
property.SetColor(1.0, 0.3882, 0.2784)              # Cor laranja
property.SetDiffuse(0.7)
property.SetSpecular(0.4)
property.SetSpecularPower(20)
property.SetOpacity(0.5)                            # Opacidade de 0.5

# Segundo ator (actor) com as mesmas propriedades e mesmo mapper
coneActor2 = vtkActor()
coneActor2.SetMapper(coneMapper)
coneActor2.SetPosition(0, 2, 0)                     # Translação no eixo Y
coneActor2.SetProperty(property)
```

Fig.2: Criação do segundo cone

2.3 Observações sobre a Opacidade

Quando a opacidade de ambos os cones foi ajustada para 0.5, observou-se que os cones se tornaram translúcidos.

Além disso, o segundo cone foi deslocado ao longo do eixo Y utilizando o método SetPosition(0, 2, 0) e, por essa razão, eles não estão sobrepostos.

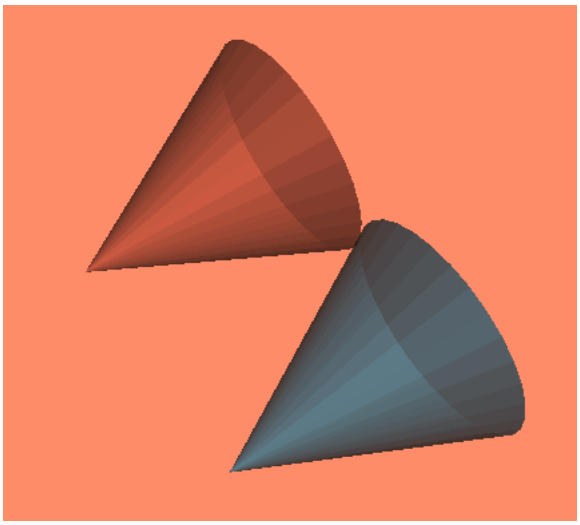

Fig.3: Opacidade e descolamento ao longo do eixo dos Y

Chapter 3

Multiple renderers

Para explorar o uso de múltiplos renderizadores em uma única janela no VTK, o exemplo cone.py foi modificado para incluir dois renderizadores distintos dentro da mesma janela. Cada renderizador foi configurado para ocupar metade da janela e apresentar perspectivas diferentes de um mesmo objeto, permitindo uma visualização simultânea a partir de diferentes ângulos.

3.1 Configuração dos Renderizadores

Dois objetos vtkRenderer foram criados e posicionados em diferentes partes da janela usando o método SetViewport(xmin, ymin, xmax, ymax). Os valores para definir os viewports foram normalizados entre 0 e 1:

- O primeiro renderizador ocupa a metade esquerda da janela (SetViewport(0, 0, 0.5, 1)).
- O segundo renderizador ocupa a metade direita da janela (SetViewport(0.5, 0, 1, 1)).

3.2 Modificação da Câmera

No segundo renderizador, a câmera inicial foi ajustada para aplicar uma rotação de 90 graus em azimute, utilizando o método Azimuth. Esse ajuste permite que os cones sejam visualizados de ângulos distintos

3.3 Alteração nas Cores de Fundo

As cores de fundo de cada renderizador foram configuradas de forma distinta para facilitar a diferenciação visual entre as áreas:

- Segundo renderizador: Cor de fundo definida como (0.2, 0.3, 0.4).

- Primeiro renderizador: Cor de fundo definida como (0.1, 0.2, 0.4).

3.4 Rotação do Cone

Foi implementada uma rotação do cone em ambas as janelas. O código foi adaptado do exemplo da primeira aula para que a rotação acontecesse simultaneamente nos dois renderizadores.

Fig.4: Esferas lado a lado

Chapter 4

Shading options

Para explorar as opções de sombreamento no VTK, o código foi modificado para substituir os cones por duas esferas. Além disso, diferentes métodos de sombreamento foram aplicados às esferas para analisar o impacto na renderização.

4.1 Configuração das Esferas

Foram criados dois objetos vtkSphereSource, cada um representando uma esfera, configurados da seguinte forma:

- **Primeira esfera:** utiliza o sombreamento padrão.
- **Segunda esfera:** teve o sombreamento configurado explicitamente para flat, Gouraud e Phong.

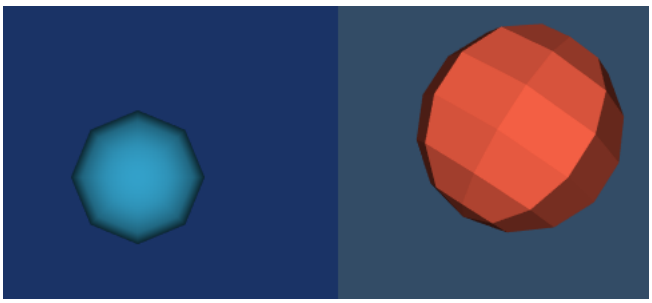

Fig.5: Flat Shading

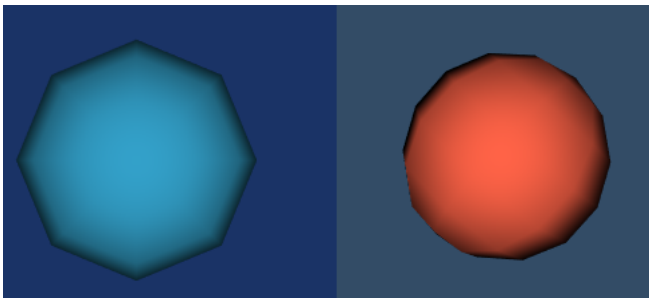

Fig.6: Gouraud Shading

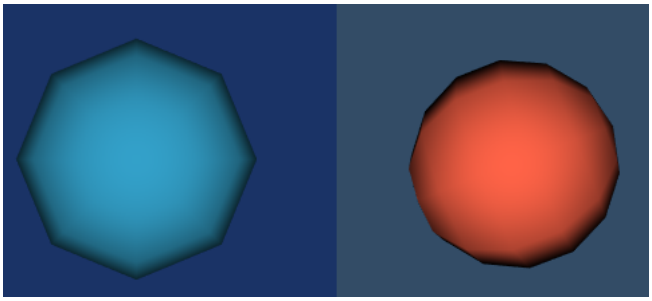

Fig.7: Phong Shading

4.2 Observações

- Com o **Flat Shading**, a segunda esfera exibe um visual mais facetado, útil para destacar a geometria subjacente.
- O **Gouraud** Shading apresenta transições suaves entre os vértices, proporcionando um visual mais uniforme e natural.
- O **Phong Shading** gera um nível adicional de suavidade e detalhe, especialmente em regiões com reflexos.

Chapter 5

Textures

Para explorar a aplicação de texturas em objetos no VTK, foi criado um programa que renderiza um plano (vtkPlaneSource) e utiliza uma imagem como textura. A classe vtkTexture foi utilizada para aplicar a textura ao plano, enquanto a imagem foi carregada utilizando a classe vtkJPEGReader.

5.1 Configuração do Plano e Textura

A classe vtkJPEGReader foi utilizada para carregar a imagem lena.JPG. Após o carregamento, a textura foi criada e configurada.

5.2 Modificação do Tamanho do Plano

O tamanho do plano foi alterado utilizando os métodos SetOrigin, SetPoint1 e SetPoint2, que definem as coordenadas de origem e os vértices do plano.

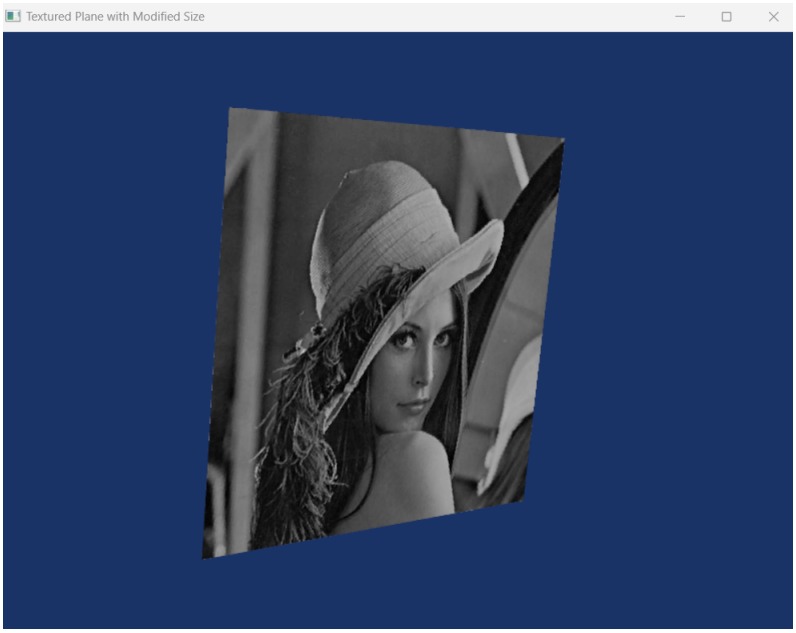

Fig.8: Imagem com textura

5.3 Observações

- **Proporção da textura:** A textura estende-se para cobrir toda a área do plano. Quando o tamanho do plano é alterado, a textura pode ser esticada ou comprimida para manter o mapeamento.
- **Distortions:** Modificações assimétricas no tamanho do plano podem causar distorções na textura, especialmente se o plano for escalado de forma não uniforme.

Chapter 6

Transformation

Agora iremos explorar a manipulação de transformações espaciais no VTK utilizando a classe vtkTransformPolyDataFilter.

6.1 Construção do Cubo com Planos Texturizados

```python
def create_textured_plane(image_path, translation, rotation):
    # Create the plane source
    plane = vtk.vtkPlaneSource()
    plane.SetXResolution(10)
    plane.SetYResolution(10)

    # Load the texture image
    reader = vtk.vtkJPEGReader()
    reader.SetFileName(image_path)

    texture = vtk.vtkTexture()
    texture.SetInputConnection(reader.GetOutputPort())

    # Define the transformation
    transform = vtk.vtkTransform()
    transform.Translate(*translation)
    transform.RotateX(rotation[0])
    transform.RotateY(rotation[1])
    transform.RotateZ(rotation[2])

    # Apply the transformation using vtkTransformPolyDataFilter
    transformFilter = vtk.vtkTransformPolyDataFilter()
    transformFilter.SetTransform(transform)
    transformFilter.SetInputConnection(plane.GetOutputPort())

    # Create the mapper
    mapper = vtk.vtkPolyDataMapper()
    mapper.SetInputConnection(transformFilter.GetOutputPort())

    # Create the actor and apply the texture
    actor = vtk.vtkActor()
    actor.SetMapper(mapper)
    actor.SetTexture(texture)

    return actor
```

Fig.9: Função para as texturas

6.2 Chamada da função

```
image_paths = [
    "./images/Im1.jpg",
    "./images/Im2.jpg",
    "./images/Im3.jpg",
    "./images/Im4.jpg",
    "./images/Im5.jpg",
    "./images/Im6.jpg"
]

# Define transformations for each plane (translation and rotation)
transformations = [
    # Front Face
    ([0, 0, 0.5], [0, 0, 0]),
    # Back Face
    ([0, 0, -0.5], [0, 180, 0]),
    # Top Face
    ([0, 0.5, 0], [90, 0, 0]),
    # Bottom Face
    ([0, -0.5, 0], [-90, 0, 0]),
    # Right Face
    ([0.5, 0, 0], [0, 90, 0]),
    # Left Face
    ([-0.5, 0, 0], [0, -90, 0])
]

# Create the renderer
renderer = vtk.vtkRenderer()
renderer.SetBackground(0.1, 0.1, 0.1)  # Dark background to highlight the cube

# Create and add each textured plane to the renderer
for i in range(6):
    actor = create_textured_plane(image_paths[i], transformations[i][0], transformations[i][1])
    renderer.AddActor(actor)
```

Fig.10: Chamada da função

6.3 Observações

- **Posicionamento do Cubo:** Cada plano foi movido e orientado utilizando combinações de rotações e translações para formar as faces do cubo.
- **Aplicação de texturas:** Seis imagens diferentes foram usadas para aplicar texturas às faces, proporcionando um visual diferenciado para cada plano.

6.4 Output

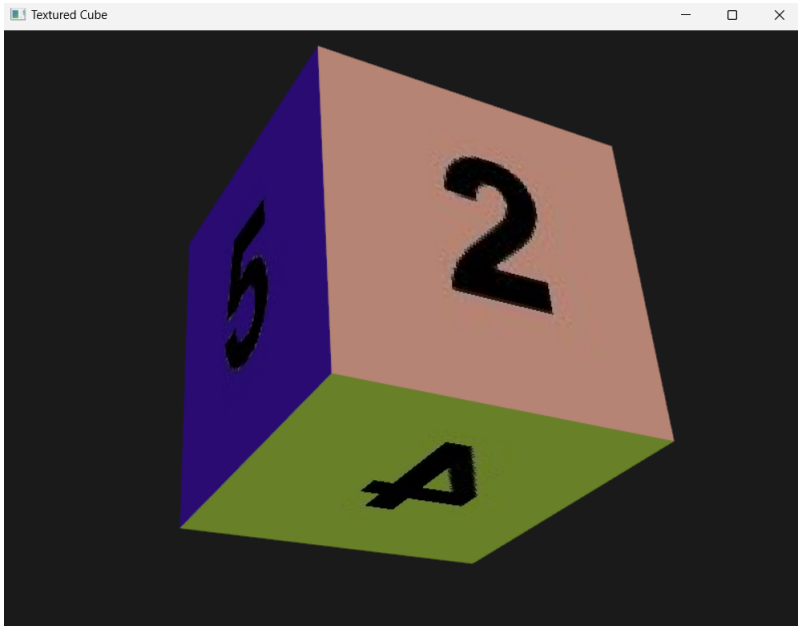

Fig.11: Cubo com textura

Chapter 7

Use of callbacks for interaction

Uma característica essencial do VTK é a possibilidade de definir funções callback, permitindo associar uma função a ser executada a um objeto sempre que um determinado evento ocorre. Essa associação é realizada por meio de um observer, que monitoriza eventos e executa a função callback correspondente.

7.1 Observações ao Executar o Programa

Ao executar o programa com o callback associado ao evento vtkCommand.AnyEvent, observamos que a função é chamada para diversos tipos de eventos ocorridos na interação, como movimentação da câmera, mudanças de janela ou atualização do renderer. No terminal é possível visualizar o nome da classe que deu "trigger" do evento, o ID do evento e a posição atual da câmera.

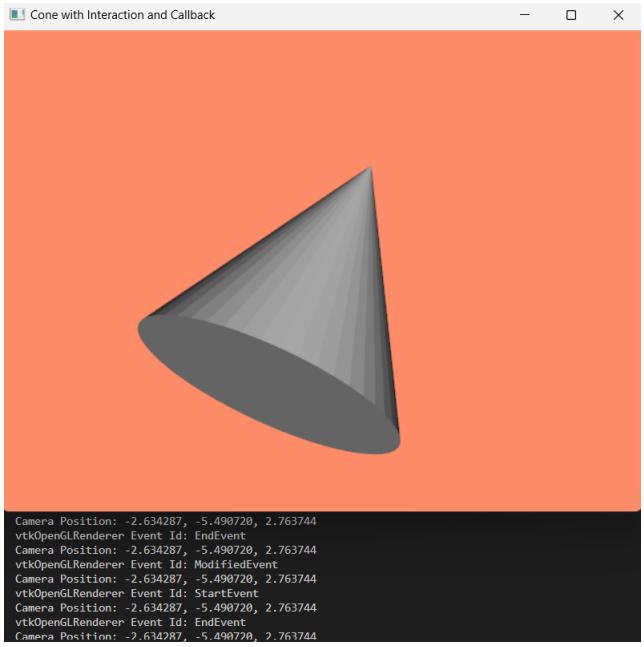

Fig.12: Função Callback associada ao Cone

Chapter 8

Contributions

Este projeto foi uma colaboração entre Rui Campos e João Ferreira. Cada membro da equipa contribuiu significativamente para as várias etapas do projeto, garantindo uma análise abrangente e o desenvolvimento robusto de exemplos práticos no VTK para explorar propriedades gráficas e interatividade em ambientes tridimensionais.

As nossas contribuições são:

- João Ferreira (103625) - 50%
- Rui Campos (103709) - 50%