Visualização de Informação

Universidade de Aveiro

João Ferreira, Rui Campos

Visualização de Informação

Lesson 1

Universidade de Aveiro

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

December 2024

Índice

1 Introduction **1**

2 First example **2**
2.1 Cone 2

3 Other primitives **4**
3.1 Esfera 4

4 Introducing interaction **6**
4.1 VTK Window Interaction 6

5 Camera Control **7**

6 Lighting **8**

7 Actor properties **9**
7.1 Alteração da Cor do Cone 9
7.2 Representação do Ator 9
7.3 Transparência 11

8 Properties and Lighting **13**

9 Contributions **15**

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 relattório, implementamos diversos exemplos práticos para compreender os conceitos de manipulação de objetos 3D, controlo de câmaras, iluminação, interação com o ambiente de renderização e ajustes nas propriedades de atores.

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, projeções e luzes, impactam a visualização final.

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

Chapter 2

First example

2.1 Cone

Após descarregar e executar o exemplo cone.py, foi possível visualizar um cone gerado pela biblioteca VTK.

Os métodos SetHeight(2) e SetRadius(1) foram usados para ajustar a altura do cone para 2 unidades e o raio da base para 1 unidade. Essas configurações alteram diretamente as dimensões do cone renderizado.

O método SetResolution é usado para definir o número de subdivisões da base do cone. Por exemplo:

SetResolution(10): Cria um cone com 10 segmentos na base, resultando em uma aparência mais angular. SetResolution(50): Cria um cone com 50 segmentos, tornando-o mais suave.

A resolução afeta tanto a aparência visual do cone quanto o desempenho de renderização.

```
coneSource = vtkConeSource()
coneSource.SetHeight(2)  # Altura do cone
coneSource.SetRadius(1)  # Raio da base do cone
coneSource.SetResolution(50)  # Resolução (subdivisões)
```

Fig.1: Height + Radius + Resolution

O método SetBackground foi utilizado para alterar a cor de fundo do renderizador para branco. Isso foi feito com o comando:

```
ren = vtkRenderer()
ren.AddActor( coneActor )
ren.SetBackground(1, 1, 1)  # Cor de fundo branca
```

Fig.2: Change background color

Os valores (1, 1, 1) correspondem às componentes RGB da cor branca no intervalo [0,1].

O método SetSize foi utilizado para definir o tamanho da janela de renderização com 300x300 pixeis. A linha adicionada foi:

```
renWin = vtkRenderWindow()
renWin.AddRenderer(ren)

renWin.SetWindowName('Cone')
renWin.SetSize(300,300)
```

Fig.3: Change window size

O tamanho padrão da janela no VTK é 300x300 pixeis. Esta é a configuração inicial utilizada se o método SetSize não for explicitamente chamado.

Chapter 3

Other primitives

3.1 Esfera

A classe vtkSphereSource foi utilizada para substituir o cone original por uma esfera. Para configurar as dimensões da esfera, foi utilizado o método SetRadius(2), definindo o raio da esfera como 2 unidades. Além disso, o método SetPhiResolution e SetThetaResolution foram utilizados para ajustar o nível de detalhe da esfera, controlando a quantidade de subdivisões nos ângulos de latitude e longitude

```
# Criação da esfera
sphereSource = vtkSphereSource()
sphereSource.SetRadius(2)  # Raio da esfera
sphereSource.SetThetaResolution(30)  # Resolução angular
sphereSource.SetPhiResolution(30)  # Resolução polar
```

Fig.4: Height + Radius + Resolution

A classe vtkCylinderSource foi utilizada para renderizar um cilindro. Os métodos SetRadius(2) e SetHeight(3) foram aplicados para configurar o raio da base e a altura do cilindro, respetivamente. O método SetResolution também foi testado para alterar o número de subdivisões na base do cilindro.

```
# Criação do cilindro
cylinderSource = vtkCylinderSource()
cylinderSource.SetRadius(2)  # Raio do cilindro
cylinderSource.SetHeight(3)  # Altura do cilindro
cylinderSource.SetResolution(50)  # Resolução angular
```

Fig.5: Change background color

Os valores (1, 1, 1) correspondem às componentes RGB da cor branca no intervalo [0,1].

O método SetSize foi utilizado para definir o tamanho da janela de renderização com 300x300 pixeis. A linha adicionada foi:

```
renWin = vtkRenderWindow()
renWin.AddRenderer(ren)

renWin.SetWindowName('Cone')
renWin.SetSize(300,300)
```

Fig.6: Change window size

O tamanho padrão da janela no VTK é 300x300 pixeis. Esta é a configuração inicial utilizada se o método SetSize não for explicitamente chamado.

Chapter 4

Introducing interaction

4.1 VTK Window Interaction

Para tornar a visualização mais intuitiva, foi adicionado um interactor ao exemplo, permitindo uma interação diretamente com a janela do VTK. O código utilizado foi o disponilibizado no GitHub.

Com esta modificação, a janela suporta interações por rato e teclado, proporcionando maior controlo sobre a visualização da forma geétrica.

Na imagem abaixo, as propriedades Pick (seleção de objetos) e Wireframe (visualização da malha do objeto) estão ativas:

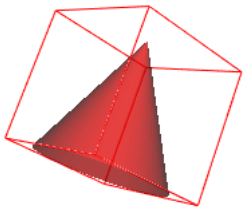

Fig.7: Wireframe + Pick

Chapter 5

Camera Control

Após a criação do renderer, foi adicionada uma nova câmara com os seguintes parâmetros iniciais:

```
cam1 = vtkCamera()
cam1.SetPosition(10, 10, 0)  # Posição inicial
cam1.SetViewUp(0, 1, 0)  # Vetor de visão para cima
ren.SetActiveCamera(cam1)  # Definir câmera no renderizador
```

Fig.8: Nova Câmara

Para obter o mesmo resultado sem criar uma nova câmara, foi utilizado o método GetActiveCamera, que altera a câmara padrão criada pelo renderer. O código foi ajustado para:

Ao alterar a câmera para modo de projeção ortográfica com o método SetParallelProjection(True) e renderizar um cubo em modo wireframe, o modelo foi exibido com proporções exatas, eliminando o efeito de perspectiva. A câmera ortográfica apresenta todas as faces do cubo com dimensões reais, úteis para análises técnicas e visualizações precisas.

```
# **Alteração para câmera padrão**
defaultCamera = ren.GetActiveCamera()
defaultCamera.SetPosition(10, 10, 0)  # Nova posição
defaultCamera.SetViewUp(0, 1, 1)  # Novo vetor de visão para cima
defaultCamera.SetParallelProjection(True)  # Ativa visão ortográfica
```

Fig.9: Câmara Padrão

Chapter 6

Lighting

O controlo da câmara foi complementado com a criação de uma nova fonte de luz, utilizando a classe vtkLight. A luz foi configurada para ter uma cor vermelha (1, 0, 0) e foi posicionada de acordo com a posição e o ponto focal da câmera ativa.

Ao adicionar a luz, a cena foi iluminada com uma tonalidade avermelhada, simulando uma fonte de luz posicionada exatamente onde a câmera está localizada. A iluminação move-se com a câmera, pois a posição e o ponto focal da luz estão ligados à mesma.

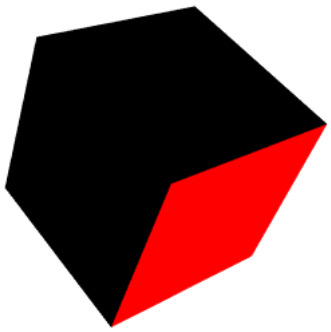

Fig.10: Lighting

Chapter 7

Actor properties

7.1 Alteração da Cor do Cone

A cor do cone foi alterada para um tom personalizado utilizando o método SetColor. Para isso, foi necessário acessar as propriedades do ator através do método GetProperty (neste caso para (1,0,0), que é vermelho).

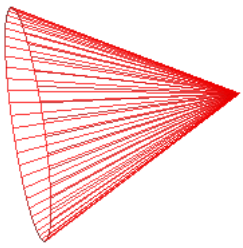

Fig.11: Alteração da cor

7.2 Representação do Ator

O método SetRepresentation foi utilizado para modificar a forma como o cone é renderizado. As opções disponíveis incluem:

- **Surface (Superfície):** Representação padrão, onde o objeto é renderizado como uma superfície sólida.

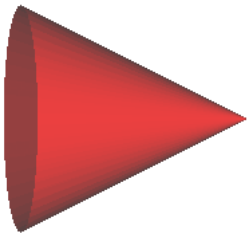

Fig.12: Lighting

- **Wireframe (Malha):** Exibe apenas as arestas do objeto, permitindo visualizar a estrutura geométrica.

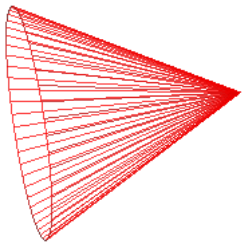

Fig.13: Wireframe

- **Points (Pontos):** Renderiza apenas os vértices do objeto.

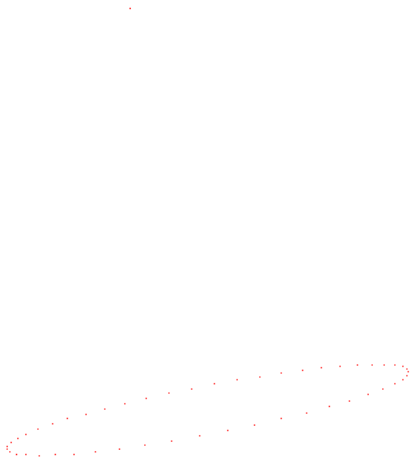

Fig.14: Pontos

7.3 Transparência

A transparência do cone foi ajustada com o método SetOpacity, utilizando os valores 0, 0.5 e 1:

- **Opacity = 1:** O objeto é opaco, com visualização completa da cor e superfície.

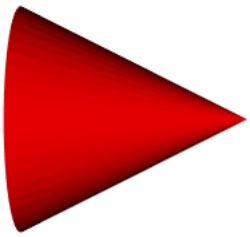

Fig.15: Opacity = 1

- **Opacity = 0.5:** O objeto torna-se translúcido, permitindo ver parcialmente através dele.

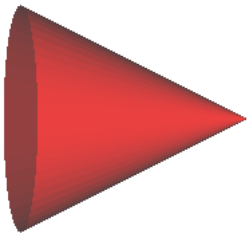

Fig.16: Opacity = 0.5

- **Opacity = 0:** O objeto desaparece completamente, não sendo visível na cena.

Fig.17: Opacity = 0

Chapter 8

Properties and Lighting

No último exercício, adicionamos várias fontes de luz ao ambiente, cada uma com uma cor específica e direcionada para a origem. Para evitar a repetição de código, implementamos uma função que ativa uma luz em uma posição e cor especificadas. As luzes utilizadas foram:

- Luz vermelha em posição $(-5, 0, 0)$.
- Luz verde em posição $(0, 0, -5)$.
- Luz azul em posição $(5, 0, 0)$.
- Luz amarela em posição $(0, 0, 5)$.

Cada uma dessas fontes de luz ilumina a cena de uma maneira diferente, criando um ambiente dinâmico, que pode ser observado ao visualizar a interação das cores.

```python
def add_light(renderer, color, position):
    """
    Adds a light to the renderer with the specified color and position.
    """
    light = vtkLight()
    light.SetColor(color)
    light.SetPosition(position)
    light.SetFocalPoint(0, 0, 0)  # Pointing towards the origin
    renderer.AddLight(light)

def add_light_sphere(renderer, color, position, radius=0.5):
    """
    Adds a sphere at the position of the light source with the specified color.
    """
    sphereSource = vtkSphereSource()
    sphereSource.SetRadius(radius)
    sphereSource.SetCenter(position)

    sphereMapper = vtkPolyDataMapper()
    sphereMapper.SetInputConnection(sphereSource.GetOutputPort())

    sphereActor = vtkActor()
    sphereActor.SetMapper(sphereMapper)

    # Set sphere color and disable lighting
    sphereActor.GetProperty().SetColor(color)
    sphereActor.GetProperty().LightingOff()  # Prevents lighting from affecting the sphere

    renderer.AddActor(sphereActor)
```

Fig.18: Code

Estas funções permitem adicionar luzes e esferas que as representam no ambiente de renderização:

- **add_light**: Cria e posiciona uma luz colorida no renderizador, com o foco no ponto de origem $(0, 0, 0)$.
- **add_light_sphere**: Adiciona uma esfera colorida na posição da luz, representando sua localização, e desativa a influência da iluminação na esfera.

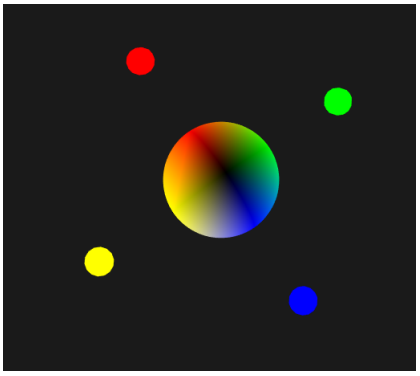

Fig.19: Final Visualization (Example 1)

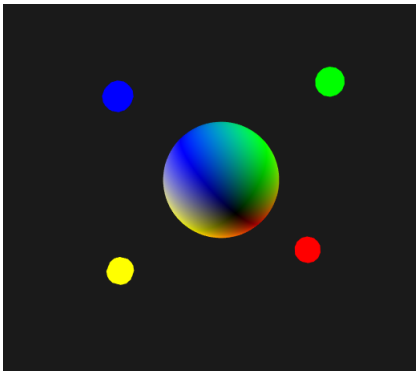

Fig.20: Final Visualization (Example 2)

Chapter 9

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.

Our Contributions are:

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