1st Lab with VTK (Visualization Tool Kit)

Information Visualization

Daniel Madureira, 107603
José Gameiro, 108840

***Resumo* - This document presents a brief description of the work developed for the first lab of the class Information Visualization, with the objective to have a first glance of the VTK framework.**

I. Introduction

The **Visualization Toolkit** (**VTK**) is an open source software system widely used for 3D computer graphics, image processing, and scientific visualization. It provides a comprehensive set of tools for creating, manipulating, and rendering complex visualizations. **VTK** supports a wide range of data types and visualization techniques, such as volume rendering, contouring , and streamline analysis.

II. Development

The lab provided by the class professor has multiple exercises that follow the following topics:

- VTK Installation;
- First VTK example, Visualization pipeline;
- VTK Interactors;
- Camera control;
- Lighting;
- Actor Properties;
- Final Exercise.

A. VTK Installation

For this exercise, it was only required to install the VTK in our local machine, and for that we used ***pip*** to install it. We used the following commands to first, create a virtual environment, activate it and then to install the VTK:

```
python3 -m venv venv
source venv/bin/activate
pip install -r requirements.txt
```

where the file specified in the last command has the VTK package for ***Python.***

B. First VTK example, Visualization pipeline

In this topic multiple exercises are present. The first one is to download and execute a file containing some ***python code*** that presents a simple example of a visualization of a cone. When executed this appears:

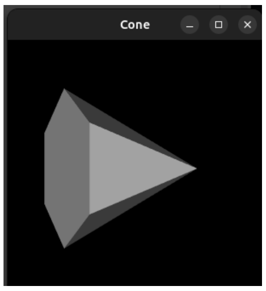

Fig. 1 - Result of the execution of the original cone.py file

It is asked to change the height and radius of the cone to 2 and 1 respectively. It is also asked to run some tests, using the *SetResolution* with different results, here are some examples:

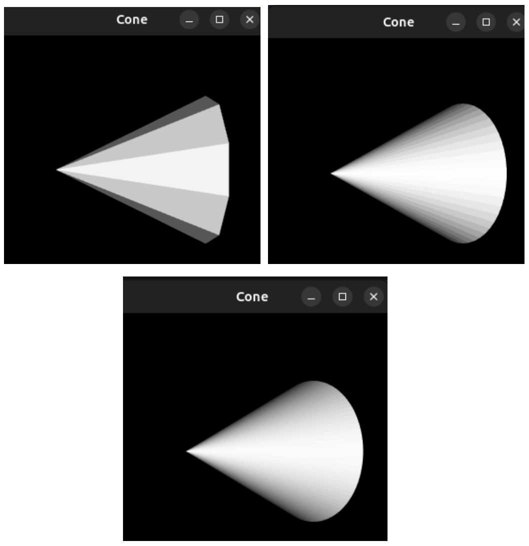

Fig. 2 - Cones with resolutions of 10, 50 and 100

We concluded that the ***SetConfiguration*** method defines the number of facets for the cone.

It also request to change the background of the window to **White**, but we decide to change it to a **Dark Purple,** using the **SetBackground method** in the renderer**:**

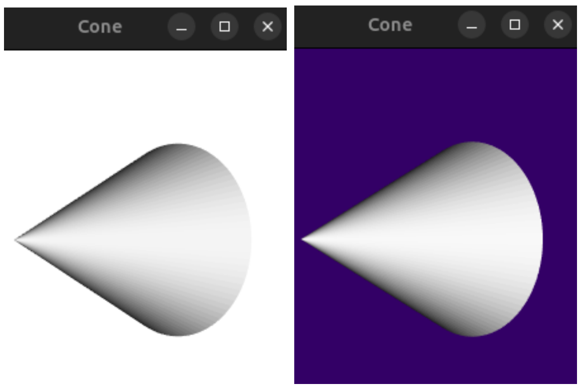

Fig. 3 - Cone with the background changed

The last exercise was to change the size of the visualization window, and indicate the default size. To accomplish this we added the following code instruction **renWin.SetSize(x, y)** where **x** and **y** represent the size of the window. We tested for the sizes **300x300**, **400x400** and **500x500** and noticed that the default window size is **300x300**.

An additional exercise was requested to include more primitives, like a sphere or a cylinder, where the sphere should have a radius of 2 and cylinder a radius and height of 2 and 3 respectively. To achieve this we created 2 functions called **render_sphere()** and **render_cilinder().** To have a better sphere we also change the values of the **Phi** and **Theta** to **100** so that the sphere would have round shape and not multiple facets. The same concept was applied to the cylinder where the **resolution** was changed to also **100.**

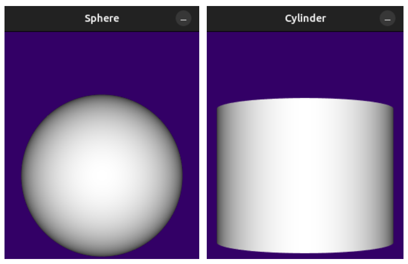

Fig. 4 - Cone with the background changed

C. *VTK Interactors*

In this section we were tasked to explore the interactor available in the VTK package. For this the following lines were added:

iren = vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
iren.Start()

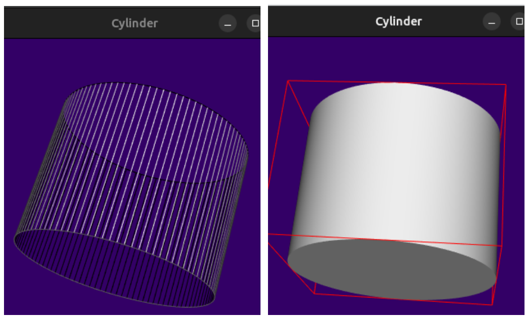

Fig. 5 - Examples of interactions with the Window Interactor

D. *Camera Control*

To explore more about Camera Control, it was asked to create a **vtkCamera** and set it at the **position (10, 10, 0)** and a **View Up Vector (0, 1, 1).** By changing the view vector, the view of the cylinder changed to a slightly oblique angle.

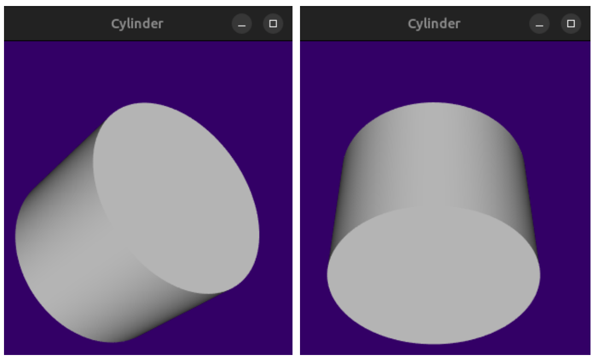

Fig. 6 - Different views of the camera, the first one with the vector (0,1,1) and the second one with the (1,0,0)

It was also asked to, instead of creating a new camera, with this line **cam1 = vtkCamera()**, retrieve the camera created by the render and change the position and the view vector, this was done by replacing the referenced line with this one **cam1 = ren.GetActiveCamera().** The last task requested was to explore the **orthographic view**, this can be done by enabling it with this line of code **cam1.SetParallelProjection(True)** and obtained the following visualizations:

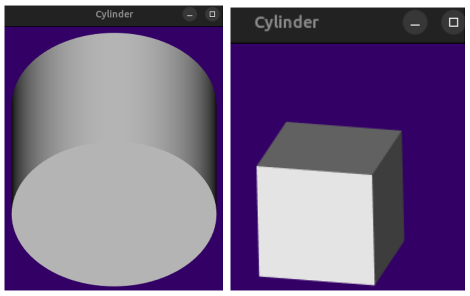

Fig. 7 - Different objects with the parallel projection on

E. *Lighting*

This topic consisted on manipulating light with VTK, and for this the following lines were added:

cam1 = ren.GetActiveCamera()
light = vtkLight()
light.SetColor(1,0,0)
light.SetFocalPoint(cam1.GetFocalPoint())
light.SetPosition(cam1.GetPosition())
ren.AddLight(light)

We obtained the following visualizations:

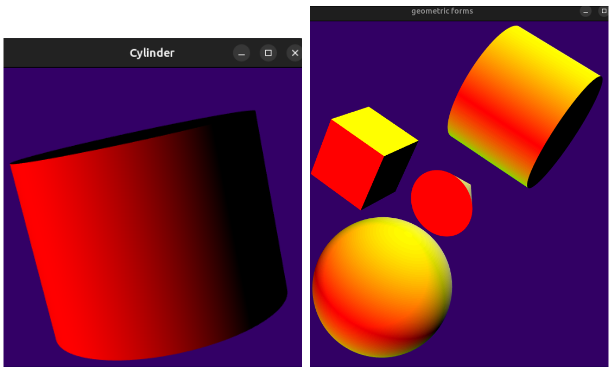

Fig. 8 - Different objects variating the color

The second image demonstrates multiple figures, with multiple lights and different colors, we applied **red, green, yellow and blue**.

F. Actor Properties

It was also requested to change the properties of an actor, for this we changed the color and opacity of the object with the following lines of code:

coneActor.GetProperty().SetColor(0.2, 0.63, 0.79)
coneActor.GeProperty().SetOpacity(1)

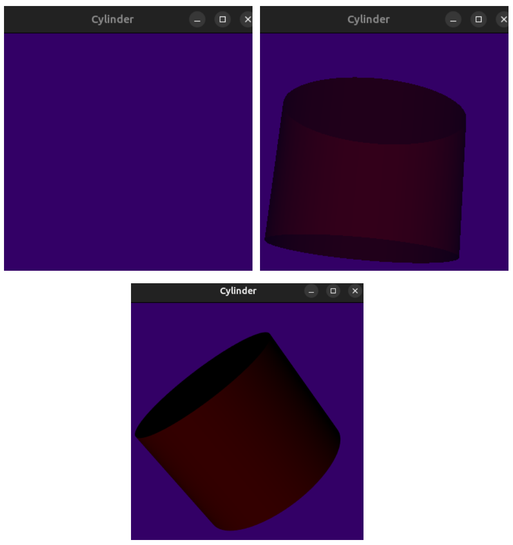

Fig. 9 - Different visualizations by varying the opacity with the values 0, 0.5 and 1

G. *Final Exercise*

The final exercise involved adding 4 lights in different positions:

- **Red** at position (-5, 0, 0)
- **Green** at position (0, 0, -5)
- **Blue** at position (5, 0, 0)
- **Yellow** at position (0, 0, 5)

and all of these positions should be pointing to the origin. Each color represents a sphere and it should be placed in the position of the light. In the center it should have an object with the different lights.

To achieve this we developed 3 main functions:

- **render_main_cone():** creates the main object that will be in the center and all the colors will be present on it. We decided to choose a cone has the object type;
- **create_light_with_sphere():** creates a light for the renderer and associates this light to a sphere. It has 3 input arguments: the **renderer**, use in the scene, the **color** of the light that will be displayed and the **position** of the light and sphere.
- **main():** the entrypoint function for this exercise, it first creates a renderer and the main object, using the **render_main_cone** function, then it adds the different lights to the renderer and finally displays the result in a render window.

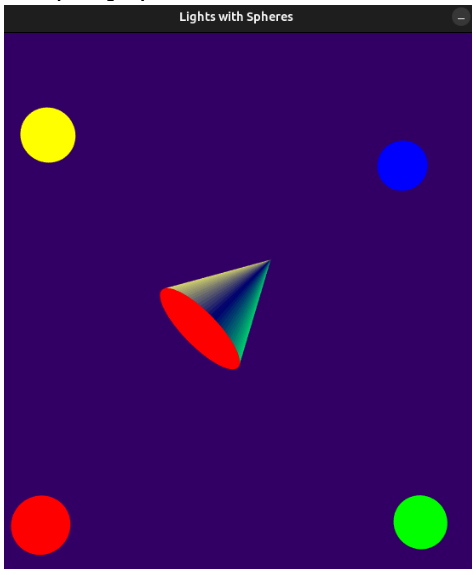

Fig. 9 - Result of the final exercise