Visual Tool Kit (VTK) - Lesson 2

Miguel Pinto
Student 107449, DETI
Aveiro University
Aveiro, Portugal
miguel.silva48@ua.pt

Tiago Figueiredo
Student 107263, DETI
Aveiro University
Aveiro, Portugal
tiago.a.figueiredo@ua.pt

I. Solved Exercises

A. Multiple Actors

In this exercise, we added two cone actors to the same scene. Both used the same mapper (which saves memory), but each had different properties, like colour, position and transparency. One actor was moved along the Y-axis to separate them in space, and the other was set to 50% opacity, showing how transparency works in overlapping objects in Figure 1.

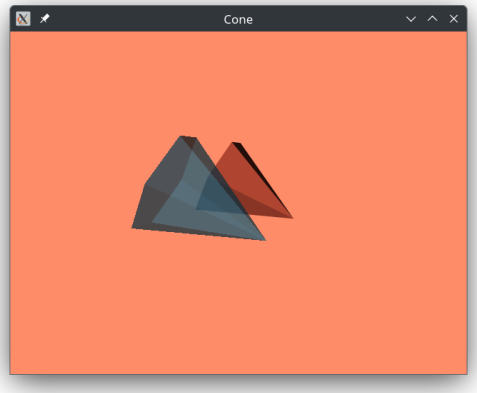

Fig. 1. Visualization of two cone actors in the same scene

B. Multiple Renderers

We created two renderers in the same window by splitting it into two viewports. Each renderer was assigned half of the window, with different background colours for clarity. The camera of the second renderer was rotated 90 degrees in azimuth, giving a distinct view of the same cone. Both renderers animated the cone simultaneously, demonstrating how multiple views of the same object can coexist in a single render window as seen in Figure 2.

C. Shading Options

Two spheres were rendered, each with a different shading method. The first sphere used flat shading, resulting in a faceted appearance where each polygon had a uniform colour. The second sphere was tested with Gouraud and Phong shading, which produced smoother appearances by interpolating light and colour across polygons. Phong shading showed the most realistic and smooth results and in the Figure 3 a comparison is seen.

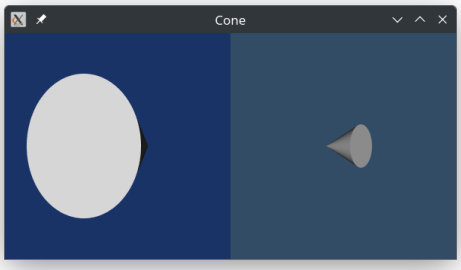

Fig. 2. Visualization of two renderers on the same window

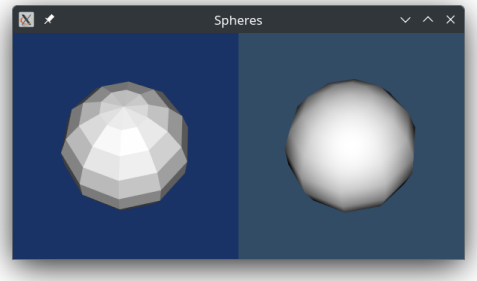

Fig. 3. Visualization of two spheres with different shading (Flat vs Phong)

Despite the shading differences, the wireframe structure of the spheres remained identical, showing that shading only affects visual lighting and not geometry which can be proven by observing Figure 4.

D. Textures

In order to map a texture (lena image) onto a plane we used the vtkPlaneSource and vtkTexture classes. By modifying the plane's dimensions with methods like SetOrigin, SetPoint1, and SetPoint2, the texture stretched or compressed to fit the new size. Changing the plane's position affected where the texture appeared in the 3D space, but the texture itself remained anchored to the plane's geometry (seen in Figure 5).

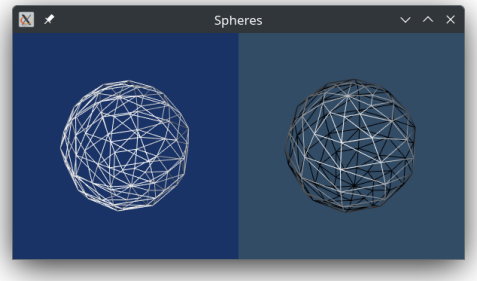

Fig. 4. Visualization of two spheres' structure with different shading (Flat vs Phong)

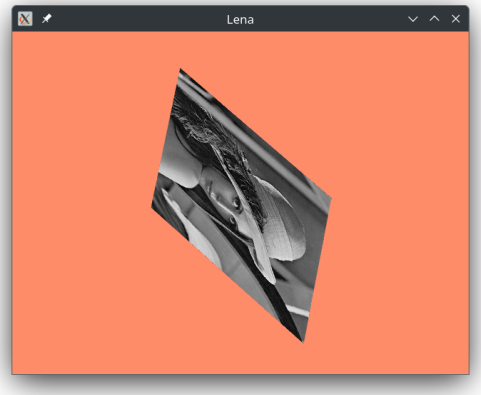

Fig. 5. Visualization of an image mapped onto a plane

E. Transformation

In this exercise, a textured cube was created using six planes (vtkPlaneSource) with unique textures mapped to each side. The vtkTransform and vtkTransformPolyDataFilter classes were used to apply translations and rotations, positioning the planes to form the cube.

To simplify the process, we created a function that takes the image file for the texture, a translation vector, and rotation angles as input, and returns an actor with the specified transformations and texture applied.

The final cube (viewable in Figure 6) was constructed by combining six planes, each with:

- **Front:** Texture aligned with the positive Z-axis.
- **Back:** Texture rotated 180° on the Y-axis.
- **Top and Bottom:** Textures rotated ±90° on the X-axis.
- **Left and Right:** Textures rotated ±90° on the Y-axis.

F. Callbacks for Interaction

In this exercise, we explored using callbacks to capture various interaction events in the VTK render window, we utilized the following types of events:

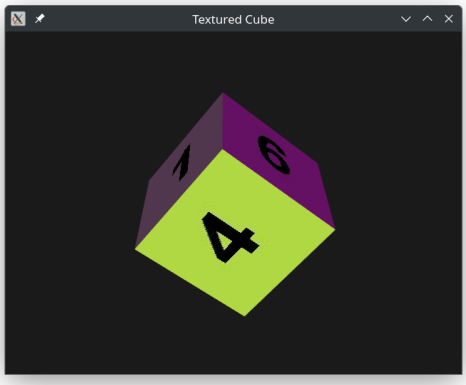

Fig. 6. Visualization of cube made of six textures

- **Start Event**: This event is triggered when the rendering process begins. It allows us to capture the start of a new render cycle.
- **End Event**: This event is triggered when the rendering process finishes. It helps to track the completion of a render cycle and can be used for post-render processing.
- **Reset Camera Event**: This event occurs when the camera is reset, often after an interaction that changes the view. It is useful for monitoring camera adjustments or resetting the scene.
- **Modified Event**: This event is fired whenever the render window or any object within the scene undergoes a modification. It is the most general event, capturing any change in the scene, such as camera movement or object updates.

These events were monitored by adding observers to the render window, and the corresponding callback function was triggered whenever one of the events occurred.

II. Conclusions

In this lesson, we explored more concepts in 3D visualization using VTK, including the creation and manipulation of multiple actors, renderers, and shading techniques, as well as applying transformations and textures to 3D objects. Additionally, we learned how to monitor different events such as rendering start, end, camera reset, and modifications.

References

[1] VTK Documentation. Available at: https://vtk.org/doc/nightly/html/
[2] Python VTK Wrapper. Available at: https://pypi.org/project/vtk/