SCIENTIFIC VISUALIZATION

Assignment 2: Volume Rendering

Miguel Moreno Gómez
Master in Innovation and Research in Informatics
Computer Graphics & Virtual Reality
Autumn Semester 2020/21

1. Introduction

The aim of this assignment is to develop a Volume Rendering application with some sort of basic lighting (i.e. Phong), an interactive histogram and an advanced function.

2. Volume Rendering Ray-casting

The volume rendering via ray-casting can be performed exclusively on GPU through a shader, although there's need for to setup some things from the OpenGL server:

To begin with, the Basic Transfer Function has to be setup: first of all, I take profit that the `volume_io` manages both the loading of the volume and the histogram array to take that array and convert it into a 1D texture. The user can interact with this TF by modifying its intensity with a slider.

The number of steps I have to do is also taken care by `volume_io` too, by using the `depth_` value and multiplying it by 1.732 (this is the distance of the diagonal between (0,0,0) and (1,1,1), the bounding box) to ensure rays can be computed from all directions.

Regarding the shader:

- For the Vertex, the only extra thing I had to add was the camera position for the fragment. In eye position, the camera is set at the origin, so I just had to multiply it by the inverse of the view matrix:

  ```
  cameraPos = ( inverse( view ) * vec4( 0,0,0, 1 ) ).xyz;
  ```

- For the Fragment, which is done via Front-to-Back, I performed these steps:
  1) Get the normalized direction between the camera and the front (which already came in via the texture coordinates). Get the accumulated colour set to zero.
  2) For each iteration:
     a) Get the value of the volume in the current position, `textureValue`. The only two important coordinates are red (as the volume is loaded in a red colour gradient) and alpha (to get transparency). The red coordinate is called "density"
     b) Get the value of the (basic) Transfer Function (TF) of the `textureValue` at the density value. Also, a cutoff for this density has been set to discard values below 0.01, as it can produce artifacts, and a value K can let the user interactively set the intensity of this TF value.
     c) Accumulate the colour as:

        ```
        acc_Colour += (1 - acc_Colour.a) * curr_Colour.a * textureValue.rrra ;
        ```

        This means that the colour accumulated takes into consideration the texture value (with all RGB coordinates using the density value, to produce a monochrome visualization), the value of the TF (`curr_Colour`) and the inverse of the accumulated transparency, which will make each points less opaque than the preceding one.

d) Move the position in the direction from 1) and by the step.
e) Perform Early Ray Termination if the conditions required (described below) are met.

3) Write into frag_color the accumulated colour obtained.

Early Ray Termination, which is performed to improve both efficiency and presentation, can happen in either of these two ways:

- When the ray is outside of the rendering box, that is, any of the `pos` coordinates is not between [0..1].
- The accumulated alpha channel is greater than 1. As this means that the colour can no longer accumulate values, it makes no sense to keep iterating through the rays.

3. Phong Shading

Once we have the accumulated colours, we can set up lighting, via Phong shading, for the volume.

Implementation-wise, it's not too different from the usual Phong shader save for one part: the normal from the diffuse component. The most common use for Phong shading (that is, using the model that comes in from the vector shader as-is) gets the normals from the vector shader trivially. Unfortunately, this is not the case here, as using the cube that contains the volume will not provide the necessary information to get the normals.

Instead, the normals are obtained from the isosurface generated from the current position: I get a the texture value for the neighbor for each x,y,z coordinate of the position, in a +- epsilon value:

```
vec3 getNormalFromVol( vec3 point )
{
    float e = 1.0f / 10.0f;

    vec3 sampledPoint;
    sampledPoint = vec3(
        texture( volume, point + vec3( e, 0, 0 ) ).r
              - texture( volume, point - vec3( e, 0, 0 ) ).r ,
        texture( volume, point + vec3( 0, e, 0 ) ).r
              - texture( volume, point - vec3( 0, e, 0 ) ).r ,
        texture( volume, point + vec3( 0, 0, e ) ).r
              - texture( volume, point - vec3( 0, 0, e ) ).r
    );
    return normalize( sampledPoint );
}
```

After normalizing, this allows me to get the normal of the point. For this reason, the shading is performed at each iteration where the colour is accumulated.

4. Histogram and advanced transfer function

For this section, I have used the QWT* library. The first thing I did was the GUI:

In the main_window class, because of the fact that QWT is external to Qt, I have had to implement the widgets manually. The steps were

1. Create a QwtPlot that will contain the histogram
2. Create said QwtPlotHistogram, and then set the size, axis scales and auto-redraw

3. Set up, for each histogram, a set of pairs of X,Y points.
4. Use the Load (Volume) function to fill-in the histogram, but only if the loading was successful. The X-axis is also set to the size of the histogram – 2 (as the first and last values are not within the [0..1] range).
5. Set up a mouse event so the user can add points to each histogram, and adding it to the set of points of the color I clicked.

On each click, the new histogram will update itself. For each point, of every colour, sorted by their position on the histogram by their X axis (I'm using a pair<X,Y> which does the sorting for me), I get the positions between 2 consecutive points P1, P2 and I calculate the new transfer function by getting the linear step to reach the P2, from P1. This then gets passed to the GLWidget, which creates the 1D texture with the braided values of each colour ($R_1 G_{1,} B_1$, $R_2 G_2 B_2$ …).

The function can be selected by a boolean function. The Alpha parameter, shared with the basic function, is not taken into account in an histogram, but in a slider (that can be used in both functions).

* As a matter of fact, I am aware that this is an allowed library; Marc Prat Masó told me, as I asked out of curiosity, that he could use it and that the library had your seal of approval.