AMCS247 - Project 3

Dan Lecocq

June 9, 2010

1 Introduction

As much modern data is inherently volumetric, the task of rendering these volumes is increasingly important. The Visualization Toolkit (VTK) provides relatively easy access to many common tasks (reading in a data file, rendering the volume, providing basic interaction with the data).

2 Motivation

We'd like to get a feel for how volume rendering and visualization works in VTK, as well as how to specify a good transfer function for this task.

3 Data

I used several of the volumetric data sets available at The Volume Library[3]. There are provided in the PVM data format, but I used a PVM to RAW [2] converter to get a bitmap out, but in order to get that to work with VTK, I had to add a header[1] to the file. Although the initial resource I looked at for guidance in this failed to get results, it gave me the idea to look at the header declaration in one of the VTK-included sets, ironProt.vtk. In short, the required header is of the format:

```
1  # vtk DataFile Version 1.0
2  <Name of File>
3
4  BINARY
5
6  DATASET STRUCTURED_POINTS
7
8  DIMENSIONS <x> <y> <z>
9  ASPECT_RATIO 1 <y/x> <z/x>
10 ORIGIN 0 0 0
```

```
11
12 POINT_DATA <x * y * z>
13 SCALARS scalars <unsigned_char|unsigned_short>
14 LOOKUP_TABLE default
```

4 Approaches

4.1 Reading the Transfer Function

To not have to recompile the program each time I want to change the transfer function, the assignment asked us to read in the transfer function from a file. As VTK uses a piecewise linear function, the file format can be as simple as

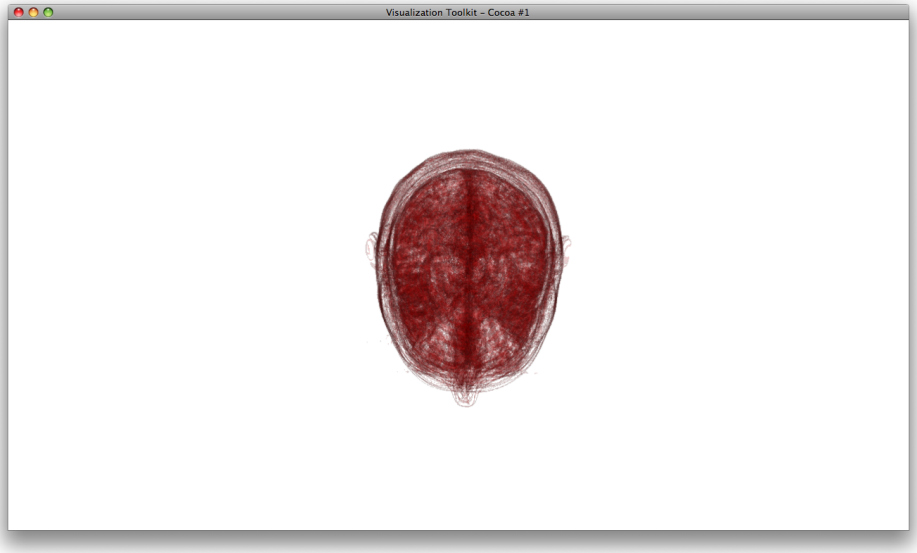
```
1 0          0.0
2 100      0.5
3 <x>      <alpha>
```

A similar file format can be used for the color transfer function, and they are read in as such:

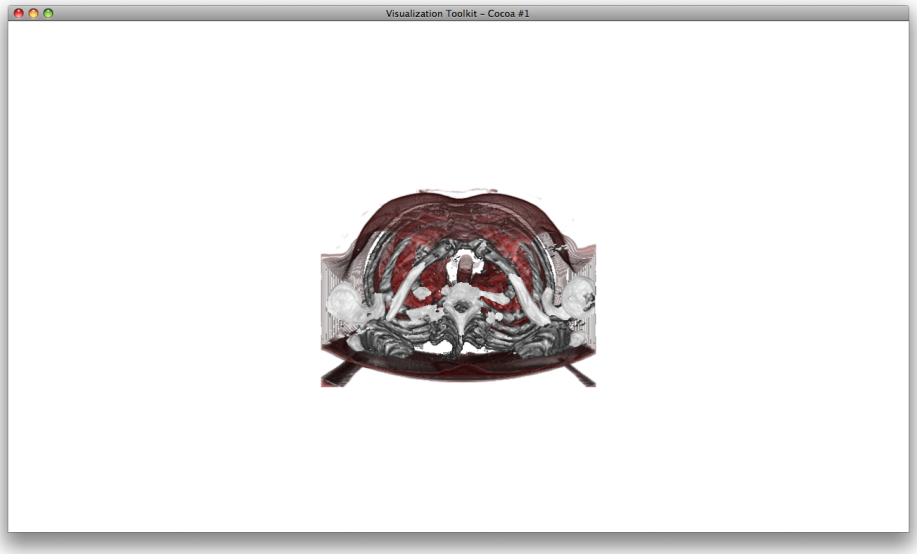
```
1  vtkPiecewiseFunction* opacity = vtkPiecewiseFunction::New();
2
3  // Open the opacity function file
4  ifstream opacity_in("opacity.f");
5  double v = 0, alpha = 0;
6  while (opacity_in >> v && opacity_in >> alpha) {
7      opacity->AddPoint(v, alpha);
8  }
9  opacity_in.close();
10
11 vtkColorTransferFunction* color = vtkColorTransferFunction::New();
12
13 // Open the color function file
14 ifstream color_in("color.f");
15 double r, g, b;
16 while (color_in >> v && color_in >> r && color_in >> g && color_in >>
       b) {
17     color->AddRGBPoint(v, r, g, b);
18 }
19 color_in.close();
```

4.2 Selecting the Transfer Function

In order to get an idea of the shape of the volume and how its density is distributed, for each, I began by setting a step function from opacity 0 to 1 at various values. For example, with the brain CT scan (Bruce.vtk) I found that skin and viscera started near density 30-40, and bone seemed to be around 80-90. Thus, to get an image at the skin around and brain inside the skull, I had opacity set to one between 30 and 70.



Others (like the chest CT-scan) required different materials (like bone and flesh) to be distinguished from one another by color. In this case, using the isovalues of boundaries I found in the previous step, I made coloring choices. Like bone might be best represented by white, and other tissue by red.



4.3 Specifying the Volume

A quick and simple change the ended up saving me a lot of time was to (again rather than compile between changes) specify the volumetric data file from the command line:

```
$> ./transfer CT-Head.vtk
```

All the code changes required to effect this were:

```
vtkStructuredPointsReader *reader = vtkStructuredPointsReader::New();
if (argc == 1) {
    // Use a default file name in case one is not specified
    reader->SetFileName("CT-Head.vtk");
} else {
    // Else use the one specified
    reader->SetFileName(argv[1]);
}
```

Granted, there is no error checking here, but for the purposes of this assignment, I believe it sufficient.

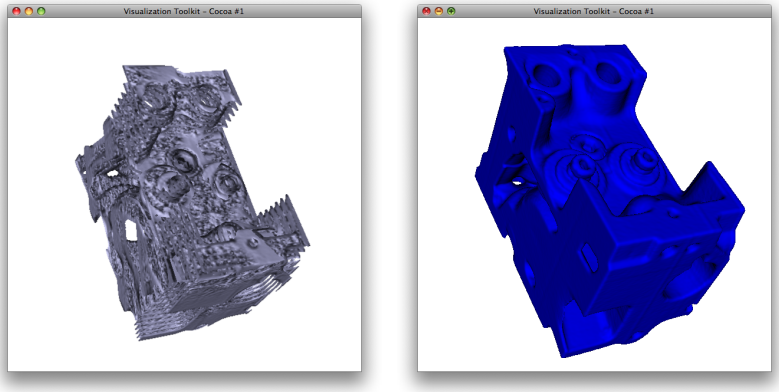
4.4 Iso-Surface

In some applications, it's more useful to see an isosurface rather than a volume render. With VTK this can be done:

```
vtkContourFilter* iso = vtkContourFilter::New();
    iso->SetInputConnection(reader->GetOutputPort());
    iso->SetValue(0,isovalue);
vtkPolyDataMapper* mapper = vtkPolyDataMapper::New();
    mapper->SetInputConnection(iso->GetOutputPort());

vtkActor *actor = vtkActor::New();
    actor->SetMapper(mapper);
    actor->GetProperty()->SetRepresentationToSurface();
    actor->GetProperty()->SetColor(1, 0.4, 0);
```

For example, the engine block is much better represented with an isosurface:



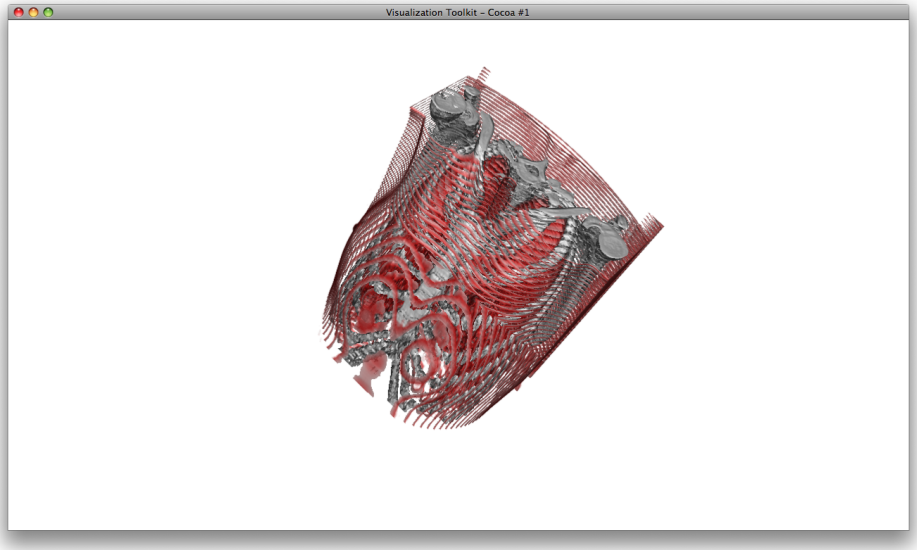
This approach can be considerably slower to render because iso-surface generation sometimes creates an enormous number of triangles. However, in many cases it leads to be results.

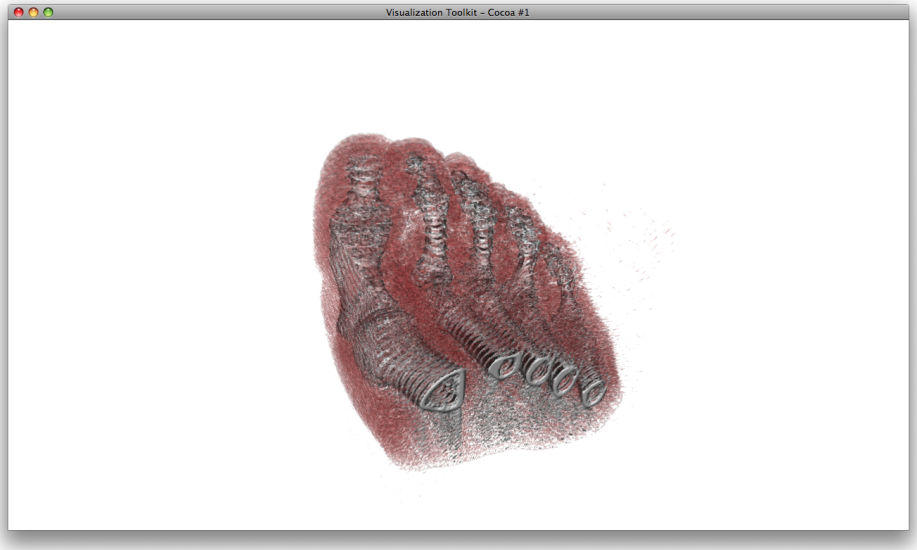
4.5 2D Texture Mapper

One of the VTK examples used a vtkVolumeTextureMapper2D to render the output, but I found the results to be, well, lackluster. They were often less readable than the ray casting approach, but perhaps I was not able to use them as well as they might be.

5 Results

Here are the results of some of the better-rendering transfer functions I used:





A References

References

[1] G-man's uber software engineering blog- how to convert a raw file to vtk. Dhttp://www.eichberger.de/2005/10/

how-to-convert-raw-file-to-vtk.html.

[2] http–www.volvis.org. http://www.volvis.org/.

[3] The volume library. http://www9.informatik.uni-erlangen.de/External/vollib/.