2D1257: Visualization
3. Algorithms and data structures

April 2, 2007

Introduction

Methods and algorithms that we will familiarize ourselves with in this lab are the following:

- Apply different types of filters on data to create visualizations
- Define your own data types and read in files with custom file formats
- Usage of implicit functions
- Evaluate the application for different visualization methods

Guides for using VTK can be found in these locations:

- Course notes
- The Visualization Toolkit Book
- http://www.vtk.org
- http://www.vtk.org/doc/release/5.0/html/

Getting started

It is a good idea to re-use the Makefile from the previous lab. The Makefile must be adjusted to this lab by changing the file names inside the Makefile.

Lab files

The lab files that you will be using for this lab are located at
`http://www.csc.kth.se/utbildning/kth/kurser/2D1257/visual07/lab3/lab3.zip`
and can be unzipped by writing `unzip lab3.zip` in the terminal. These files are:

- `ImpFunc.cc` This contains the skeleton code for the implicit function definition.
- `TraceReader.hh` Contains the skeleton code for your own data reader.
- `mob.data` This file contains the data which you will try to read using your data reader.

Note on VTK

VTK uses *forward declaration* (declaring a type for the purpose of providing an interface or protocol for the programmer - that's you, without writing what the type does). This can cause compiler errors if one tries to cast a type that is not *fully declared*. To prevent this from occurring, one can include the header file which contains the full declaration of the type.

In Short, always remember to include the header file of the type that you will use in your program, i.e., if you use `vtkLight`, you must add `#include <vtkLight.h>` to your program.

How to do this lab

In this lab, you are asked to complete only 1 of 2 tasks. You may choose which task you wish to complete. The two tasks that you may choose from are:

- Studying the wave function of a hydrogen atom using isosurfaces and implicit functions, writing and reading to and from file. (See Task 1)
- Studying the motion of an electron using vector fields and streamlines, reading from file. (See Task 2)

Remember to add interactivity to the visualization like you did in the previous lab.

1 Task 1: Wave Function of a Hydrogen Atom

1.1 What you will be doing in this exercise

In this exercise, we will use the wave function of a hydrogen atom in the state 2p which gives us the probability of finding an electron at each position of the atom. To be able to visualize this, we will use an isosurface. This is where the implicit function comes in. We create an implicit function for the purpose of visualizing the wave function using VTK's isosurface.

1.2 The wave function

The wave function can be written in the following form using the spherical coordinate system.

$$\begin{aligned} F(\phi,\ \theta,\ r) &= |Y(\phi,\ \theta)R(r)|^2 \\ Y(\phi,\ \theta) &= \pm\sqrt{\frac{15}{2\pi}}\,(\sin\phi\cos\phi)\,e^{\pm i\theta} \\ R(r) &= \left|\left(\frac{Z}{2a_0}\right)^{\frac{3}{2}}\frac{Zr}{\sqrt{3}a_0}e^{\frac{-Zr}{2a_0}}\right| \end{aligned} \tag{1}$$

In our case, we can set:

$$Z = 1, \quad a_0 = 1$$

We have a set of equations which allow us to go between the Cartesian coordinate system and the spherical coordinate system (More info can be found at the wikipedia website). To go from the

spherical coordinate system to the Cartesian coordinate system, use:

$$x = r\sin\phi\cos\theta$$
$$y = r\sin\phi\sin\theta$$
$$z = r\cos\phi$$

To go from the Cartesian coordinate system to the spherical coordinate system, use:

$$r = \sqrt{x^2 + y^2 + z^2}$$
$$\phi = \cos^{-1}(z/\sqrt{x^2 + y^2 + z^2})$$
$$\theta = \tan^{-1}(y/x)$$

Note that we do not support complex numbers. This means that we have to change the evaluation of $Y(\phi, \theta)$ to avoid going into complex numbers. There are a few ways of doing this, one way is to consider a complex number in terms of a vector.

$$z = a + bi$$
Can be written as a vector
$$z = (a, b)^\top$$

When we do this, we must remember that when we take the absolute value $|z|$ of our complex number, we actually calculate the length of the number in the Argand Diagram. Thus, when we calculate the absolute value of our vector z, we end up calculating its length instead. So now we can re-write the functions Y and R from above (note we can remove the $\pm$ because everything is squared in the end).

$$F(\phi,\ \theta,\ r) = \left|\sqrt{Y(\phi,\ \theta)^2_{Real} + Y(\phi,\ \theta)^2_{Imaginary}} \cdot R(r)\right|^2$$
$$Y(\phi,\ \theta)_{Real} = \sqrt{\frac{15}{2\pi}}\,(\sin\phi\cos\phi)\,(\cos\theta)$$
$$Y(\phi,\ \theta)_{Imaginary} = \sqrt{\frac{15}{2\pi}}\,(\sin\phi\cos\phi)\,(\sin\theta)$$
$$R(r) = \left|\left(\frac{1}{2}\right)^{\frac{3}{2}} \frac{r}{\sqrt{3}} e^{\frac{-r}{2}}\right|$$

1.3 Implicit function

To create an implicit function you have to define a subclass to `vtkImplicitFuncion`. Copy the content from the file `ImpFunc.cc` to your program and implement the functions. The following code shows you how to use your functions to create the iso surfaces. Iso surfaces should then be your input to a `vtkPolyDataMapper`.

```
...
ImpFunc * function = ImpFunc::New();
vtkSampleFunction * sample = vtkSampleFunction::New();
sample->SetImplicitFunction(function);
sample->SetModelBounds(...);
sample->SetSampleDimensions(...);
sample->ComputeNormalsOff();
```

```
vtkContourFilter * iso = vtkContourFilter::New();
iso->SetInputConnection(sample->GetOutputPort());
iso->SetValue(...);
...
```

Fill in the rest and set the values for the three surfaces. VTK will not sort the polygons, so the blending of semi-transparent rasters performed by the graphics hardware will result in artifacts, making it look like elements furthest away are in front of the elements that are close to the viewer.

1.4 What to do

You shall implement the wave function and then let it be sampled in a Cartesian coordinate system with the resolution $32 \times 32 \times 32$. The origin should be in the centre of the volume with range $[-6, 6]$ in each dimension.

The data that this function produces shall be visualized as three iso surfaces; the iso-values are 0.001, 0.002 and 0.003. Adjust the colour ramp with `vtkPolyDataMapper` to work within the interval [0.0008, 0.004] (`SetScalarRange` is part of `vtkPolyDataMapper2D` but also works for `vtkPolyDataMapper`). Define the surfaces to be transparent. Do that with the class `vtkProperty` through `vtkActor` by using the function `GetProperty()`.

The volume that you have created should now be saved to disk. Write some code in you program which writes the data to file. Make a modified version of the program that reads in a file instead of calculating it from the data again. Look at the file that you created and take a look at how it is structured. You may feel inclined to use the following VTK types for reading and writing data: `vtkStructuredPointsReader` and `vtkStructuredPointsWriter`.

1.5 How do I use sin and cos in C++

To use trigonometric functions as well as powers and square roots, you have to include the C math header.

```
#include <cmath>
```

This will give you access to the following functions:

```
pow(double, double)     // x^1.5  can be written as pow(x,1.5)
sqrt(double)            // calculates the square root
cos(double)             // trig functions
sin(double)
tan(double)
acos(double)            // inverse trig functions
asin(double)
atan(double)
...                     // there are more functions, but you do not
have to worry about them
```

2 Task 2: Motion of an Electron

2.1 Data representation

In this exercise, you shall write a program which reads in an external file containing data (more specifically `mob.data`). The data describes the motion of an electron moving in a magnetic field. The data consist of the position vector, p, and velocity vector, v. Each row of the file consists of p_x, p_y,

p_z, v_x, v_y, v_z. The class `TraceReader` that you shall implement is defined in `TraceReader.hh`. You shall create a VTK data structure containing the point positions, the square of the x velocity component, and the cell structure of the points. The VTK classes that you will need to build this are the following:

- `vtkPolyData`, the main object that will contain all the other objects
- `vtkCellArray`, defines the order of the cells and what points should belong to which cell
- `vtkPoints`, defines the position of a point
- `vtkFloatArray`, defines an array with the type float. The number of values in each field in the array is specified by `SetNumberOfComponents()`.

The cell array defines how the points in the data set are connected into primitives. This is an attribute of the `vtkPolyData` object. The positions of the points are associated with the base type of the object, the `vtkPointSet`. Thus, the functions used to assign the positions of the data points are found in the documentation of this class. The data of the points, i.e. the square of the x velocity component in this case, is associated with another base type, the `vtkDataSet` class.

Put your code that reads the data in the `GetOutput` function of your reader class. Be careful so that the data is only fetched from the file once and not every time the VTK pipeline system calls your function.

```
FILE * dataFile = fopen(fileName, "r");
float pos[3];
float vel[3];
while (fscanf(dataFile, " %f %f %f %f %f %f \n",
              pos, pos+1, pos+2, vel, vel+1, vel+2) != EOF) {
  // Create elements in dataset
  ...
}
fclose(dataFile);
```

Now build a small program that reads in the dataset from the file `mob.data`. You can easily draw the data with `vtkPolyDataMapper`. Set the scalar range in the mapper to [0.0, 0.01] so that the points are coloured correctly using the v_x^2 values.

Go back to you reader class. Modify it to connect all the points into one single long line. To generate lines, the cell array contains series of points in each cell. If all points are in a single line then all points must be in the same cell.

2.2 Vector fields

Above, you created a 3D volume from an implicit function. Another method for creating image data is to directly build the data-cluster in the code. You shall now define the magnetic field that the particle was moving in and visualize it *together* with the particle trace.

This you do by first creating a structure with the help of `vtkStructuredPoints`. Set the dimension to $64 \times 64 \times 64$, and the origo of the volume to $(-1, -1, -1)$. Set the spacing to $1/32$ (spacing is a feature of the volume). After this, create a triple loop to calculate the magnetic field in each point location and assign vectors to a `vtkFloatArray` structure.

The single 1D index for indexing the array is calculated as follows:

$$i = k + 64(l + 64m)$$

where k, l and m correspond to the point in x, y and z. Coordinate z is therefore calculated as:

$$z = -1 + \frac{1}{32}m$$

(since spacing is 1/32). The magnetic field B is defined as

$$B = (b_x, b_y, b_z)$$

where the elements are calculated as follows:

$$b_x = \tanh(z), b_y = 0.188, b_z = 0.1$$

Fetch after this the point-data from the structured points and assign the vectors to this.

[Note: A common mistake is to do integer division when floating point division is required.]

To visualize this data you should first use a hedge-hog plot. This is simply done by using the `vtkHedgeHog` class. Play around to find a suitable scale factor.

2.3 Stream lines and stream tubes

We will now create some streamlines to visualize the magnetic field B, together with the particle trace. Use `vtkStreamLine` to do that. Streamline requires you to specify a set of points that the lines should pass through. One way of doing this is by defining a plane and then choosing certain points on this plane.

Use `vtkPlaneSource` and set its origin to (-1,-1,0). Set point 1 as (1,1,0) and point 2 as (1,0,0). Set the resolution to 5 in each dimension. You are encouraged to experiment with this! Use this plane as a source for the streamlines. The data to the streamlines is the magnetic field we created before. Investigate suitable values for step length, integration step length as well as maximum propagation time. To make the streamlines appear on both sides of the plane you have to make sure that the integration happens in both directions. Now we have both lines to visualize the magnetic field as well as the path of the particle.

Now use `vtkTubeFilter` to create 3D-tubes instead of lines. The input data to the tube filter is of course the streamlines.