DATA VISUALIZATION USING OpenGL

Mary Slocum '13G*
Graduate Student, M.S. Program in Computer Science, Rivier University

Abstract

Data mining answers our desire to represent vast amounts of different data in a way that users can best understand the results. Using Open Graphics Library (OpenGL), we can create the computer graphics for the lines, text, and shapes of a graph for visualization. This process is beneficial for all students from computer science to nursing students when representing data.

The motivation for creating the data visualization using OpenGL was inspired by watching Hans Rosling's TED talk on "No more boring data"[1]. Using OpenGL, the goal to is to show how to create a similar graph with different data. The data can vary from showing the progress of countries or states with differing statistics.

1 Introduction

OpenGL is an open-source and cross-platform library of functions used to create computer graphics [2]. This can be easily used with C++ on Windows as well as under Linux or Mac OS X. In addition, there are other available libraries such as the commonly used OpenGL Utility Toolkit (GLUT). Though GLUT allows easy creation of an OpenGL windows application using C++, it is not open source [3].

Under Windows, there are standard header (included), library, and dynamic link library (DLL) files available and downloadable for both 32 and 64-bit development. All of the OpenGL functions start with 'gl' while the GLUT functions have the prefix of 'glut'. The OpenGL functions have a suffix determining if the parameters will be double or float values. For example, both glColor3d and glColor3f are functions specifying the color, but glColor3d takes double values for parameters while glColor3f takes float-point values.

2 Implementation and Data

Given the emphasis on President Obama's data crunchers in the last election, we selected four sets of data to visualize in one graph using OpenGL for representing past election results. Each circle on the graph will represent a state with the color showing what political party the state voted for in that election year. The circle's size indicates the state's population as compared to the other states (with the smallest state being shown by the smallest circle) [4]. For the x and y axes, we use per capita income [5] and children in poverty respectively. The childhood poverty data represents the number of children under age 18 living in families with incomes below the federal poverty level [6]. As a proof of concept, data for only the past three presidential election years (2000, 2004, and 2008) will be used [7]. This animation shows the progression of the statistics over the years.

Real data is collected from various web sites for each state per election year. The comma-separated value (CSV) file compiled, <ElectionYear>Sorted.csv, of the real data is sorted by state determining the size of the circle based on this ascending order. Standard C++ stream functions are used to read this file into a local table variable for use by OpenGL. See Figure 1 for a sample of the CSV file for the

ISSN 1559-9388 (online version), ISSN 1559-9396 (CD-ROM version).

election year 2000 being used where the first row contains the column names and subsequent rows have the data.

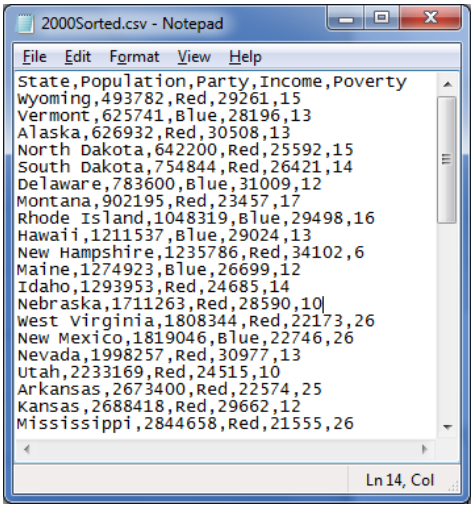

Figure 1. Sorted data (in ascending order based on population) in CSV file for the Year 2000.

Also, a standard (hardcoded) map data type is used to contain the states and their corresponding abbreviations for the text on the circles.

3 Visualization with OpenGL

To create the graph, the drawing components of text, lines, and circles must be generated. For all of the components, color can be set individually using a function such as:

```
glColor3d(0, 0, 1);         // Set current color to blue
```

With the glColor function, each parameter contains the values for red, green, and blue (RGB) respectively.

For all graphic primitives (such as lines and circles), the glBegin/glEnd delimiters are used to indicate what the vertices pertain to. For example, if using glBegin(GL_QUADS) , then the four vertices will represent the corners of the quadrilateral [8].

With OpenGL, it is important to always calculate and set the position of the objects correctly. This is typically done by a function, such as `glRasterPos2f`. In addtion, the glTranslatef function can be used to translate the current matrix for positioning.

3.1 Lines

To create the graph grid, generation of both horizontal and vertical lines is required. For the vertical lines, the following code needs to be executed based on the number of lines required specifying the start and end points of each line using `glVertex2d`:

```
glBegin (GL_LINES);
   // Y-Axis
   glVertex2d(x+i, y);   // Specify line segment geometry
   glVertex2d(x+i, -y);
glEnd();
```

The horizontal lines would also be created in a loop for the number of lines required with the necessary x and y coordinates as shown below:

```
glBegin (GL_LINES);
   // X-Axis
   glVertex2d(-x, y-i);
   glVertex2d(x, y-i);
glEnd();
```

3.2 Text

In the graph, we display text for the x and y axes, the title, and the state abbreviations on each circle. Using OpenGL with GLUT, this can be done in two ways: glutBitmapCharacter and glutStrokeCharacter. With glutBitmapCharacter, this function is called multiple times for each character in the string specifying the font to be used such as:

```
glutBitmapCharacter(GLUT_BITMAP_HELVETICA_10, input[j]);
```

This `glutBitmapCharacter` call is done after specifying the starting position of text using the function `glRasterPos2f`. Then, the glutBitmapCharacter function is used to create the horizontal text for the title and x-axis.

For the rotated text (“Children in Poverty”) on the y-axis, we need to use the function `glutStrokeCharacter` for our character rendering instead of `glutBitmapCharacter` as this fuction does not support rotation. First, we would precede this call with a `glRotated` to specify the rotation amount. For example, to display vertical text, we rotate by 90 degrees as in the below sample code:

```
glRotated (90, 0, 0, 1);  // Rotate text to be vertical
glutStrokeCharacter(GLUT_STROKE_ROMAN, input[i]);
```

This code will also need positioning first which could be done by `glTranslatef`. If using `glRotate` or `glTranslate`, one can optionally use `glPushMatrix/glPopMatrix` for these transformations to set the current matrix and then pop it.

3.3 Circles

Next, either a red or blue circle is created based on the party receiving the majority of votes that election year. To draw a circle, we create many very small triangles varying the color slightly each time to create shading. Sample code for creating a red shaded circle might look something like:

```
glRasterPos2f(x, y); // Set position of circle

// Draw a circle with blended red vertices using triangles
const float step = PI / 16;
glBegin( GL_TRIANGLE_FAN );
glColor3f( 1.0f, 0.0f, 0.0f );  // Set  initial red color
for ( float angle = 0.0f; angle < ( 2.0f * PI ); angle += step )
{
      float fSin = sinf(angle);
      float fCos = cosf(angle);
      glColor3f( ( fCos + 1.0f ) * 0.7f, 0.0f , 0.0f);  // Red blends

      // Set next position
      glVertex2f( x + (radius * fSin), y + (radius * fCos) );
}
glEnd();
```

3.3.1 Text on circles

Finally, circle text is added using glutBitmapCharacter in a manner similar to the x-axis and title text. The text will use a smaller font and a different color to make it more visible. As the text is created after the circle is drawn, the `glDisable (GL_DEPTH_TEST)` and `glEnable (GL_DEPTH_TEST)` must be around the loop of calls to glutBitmapCharacter in order for depth comparisons to be done and for updates to the depth buffer to occur.

The following sample code will display white horizontal text in the center of the circle:

```
glDisable (GL_DEPTH_TEST);
glColor3d (1, 1, 1);            // Set current color to white
glRasterPos2f(x-(radius/2), y); // Set position of text in center
// Get state abbreviation from map
string input = m_stateAbbrev[dataHelper.getState(i)];
int len = input.length();
// Use Bitmap Text for horizontal
for (int i = 0; i < len; i++)
{
   glutBitmapCharacter(GLUT_BITMAP_HELVETICA_10, input[i]);
}
glEnable (GL_DEPTH_TEST);
```

3.4 Graphs

After combining all of these drawing components, the various graphs are created based on input data from the CSV files. The figures below were created by using OpenGL to show the data in graph format for each of the election years.

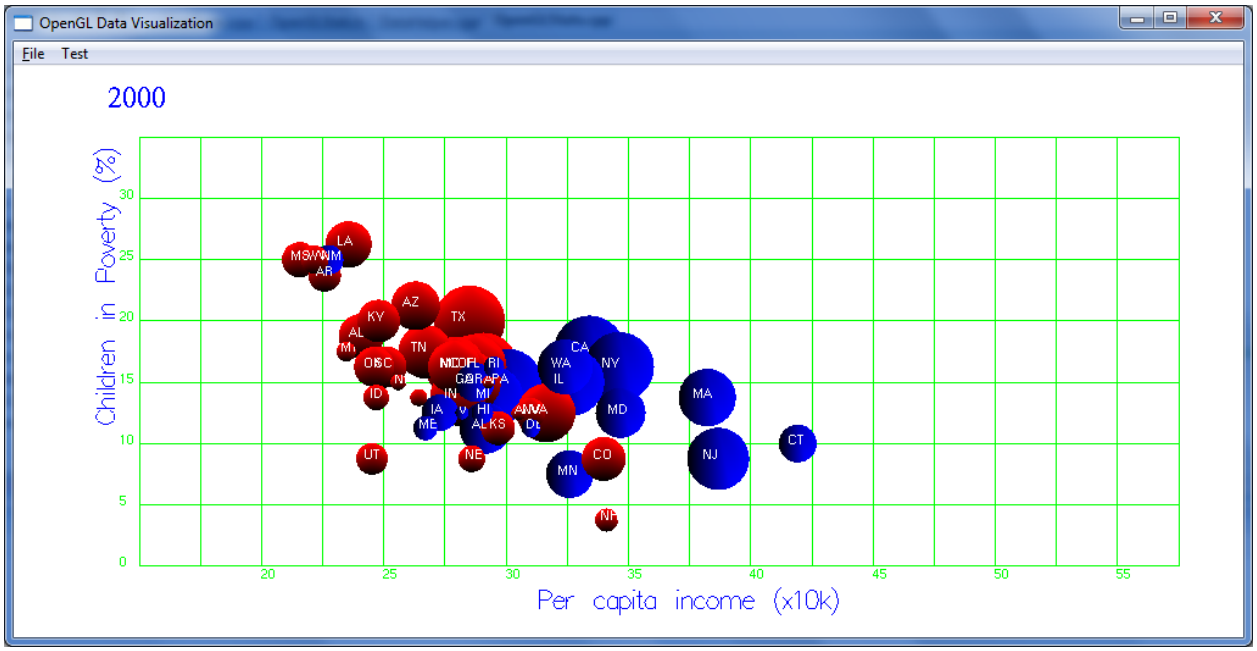

Figure 2. Graph for Year 2000.

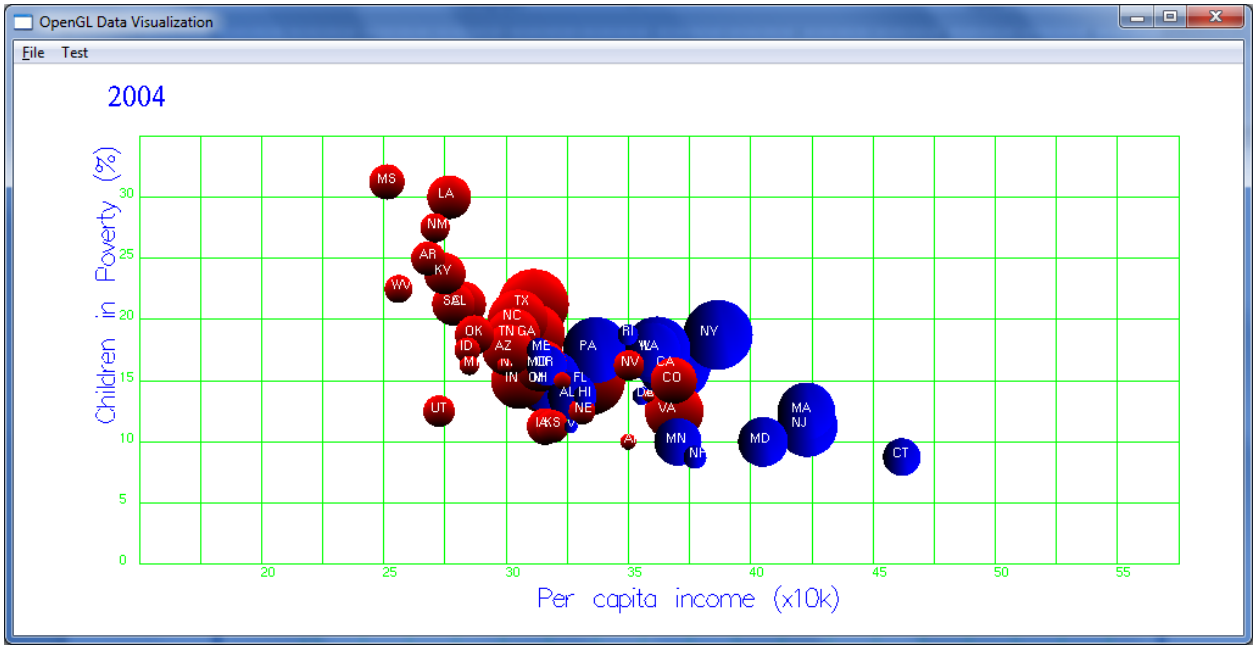

Figure 3. Graph for Year 2004.

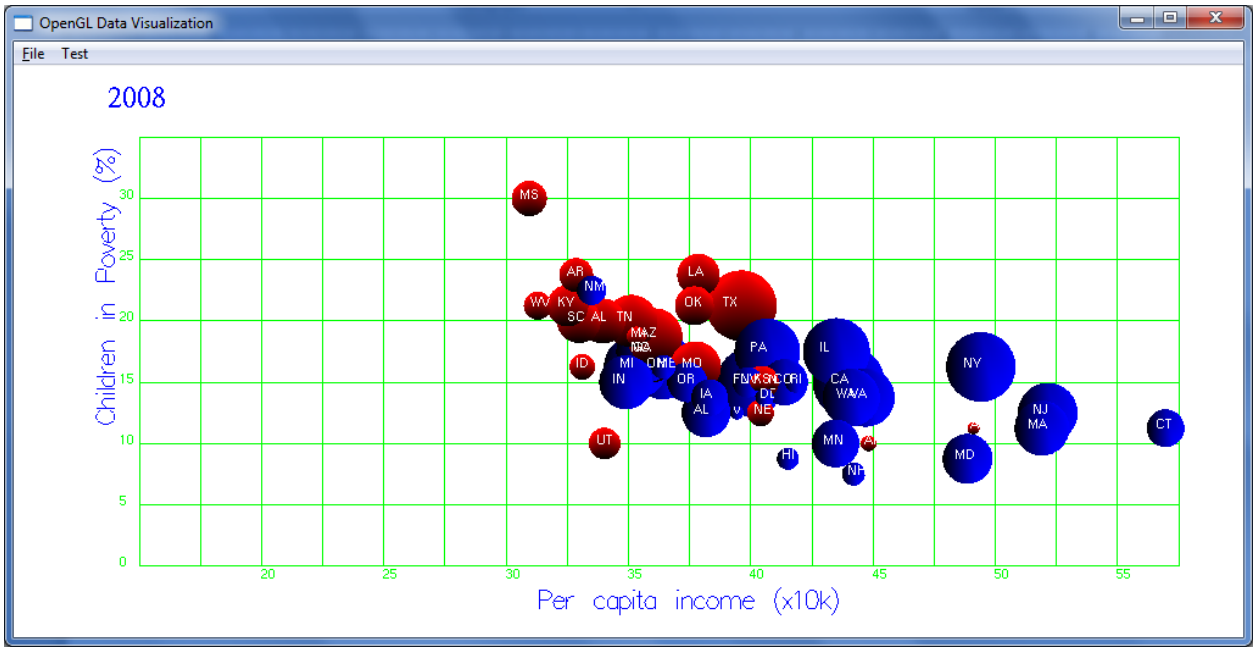

Figure 4. Graph for Year 2008.

One interesting thing we can see from the graphs is that Connecticut (CT) consistently has the highest average per capita income and has voted for the Democratic candidate in the past three presidential election years. In contrast, Mississippi (MS) has the lowest per capita income and voted Republican in the same election years. In 2008, most of the states with over 17% of children in poverty voted Republican while those states with under 17% of children in the poverty level selected the Democratic candidate.

4 Conclusion

Using these techniques with OpenGL, we can also display different data as per individual needs. A medical student might desire to show statistics over the years in different states for various diseases. For police, this could be used to show trends of where and what crimes are occurring in a particular area.

Using OpenGL, we have seen that it is relatively straight forward to create a basic graph. Of course, this representation can be enhanced in many different ways such as adding the ability to look at only certain sections of the country (such as focusing on the northeast) or increasing the gap in the x-axis so that many states do not overlap making the results clearer. In addition, we can use other sets of data for the x and y axes along with adding more election years to better show the trends occurring.

References

1. Rosling, H. TED talk on “No more boring data”. Retrieved February 16, 2013, from http://www.youtube.com/watch?v=hVimVzgtD6w
2. Hearn, D. D., Baker, M. P., and Carithers, W. *Computer Graphics with OpenGL*, 4th edit., Prentice Hall, 2011, p. 40.
3. “GLUT – The OpenGL Utility Toolkit”. Retrieved February 16, 2013, from http://www.opengl.org/resources/libraries/glut/

4. U.S. Population by State (size of circle): 2008 & 2004. Retrieved February 16, 2013, from http://www.infoplease.com/ipa/A0004986.html
5. Per Capita Income by State (x-axis). Retrieved February 16, 2013, from http://bber.unm.edu/econ/us-pci.htm
6. Children in poverty (Percent) (y-axis). Retrieved February 16, 2013, from http://datacenter.kidscount.org/data/acrossstates/Rankings.aspx?loct=2&by=a&order=a&ind=43&dtm=322&tf=35
7. Presidential results (color of circle: red or blue). Retrieved February 16, 2013, from http://en.wikipedia.org/wiki/United_States_presidential_election,_2008; http://en.wikipedia.org/wiki/United_States_presidential_election,_2004; and http://en.wikipedia.org/wiki/United_States_presidential_election,_2000
8. glBegin. Retrieved February 16, 2013, from http://msdn.microsoft.com/en-us/library/windows/desktop/dd318361(v=vs.85).aspx

* **MARY SLOCUM** is a Computer Science major expecting to complete her Masters at Rivier University in spring 2013. She resides in Nashua with her husband and two daughters. She is originally from New York where she worked in NYC before relocating to New England.