TNM048 – Information Visualization
1. Introduction to C# and GAV

January 22, 2007

1 Introduction

This lab exercise will introduce the concept of component programming and show how high-level programming can be used to produce advanced and interactive visualization applications. The NVIS-developed library GeoAnalytics Visualization Framework (GAV) will be used in conjunction with C# .NET. After completing the lab exercise, you should be able to understand:

- the basics of C# and the basic functionality of the .NET Framework
- how to create a standard executable
- how to use third party components in order to add functionality to an application
- how the data flow is managed in a GAV application
- how a basic chart application is constructed

To pass this exercise, functional code as well as your explanations of how it works are required. The laboratory exercise should be performed individually or in a group of maximum two students. After completing all tasks, present your results to the laboratory assistant.

2 .NET Framework

The Microsoft .NET Framework is a software component which provides pre-coded solutions to common program requirements and handles the execution of programs targeting the framework. The pre-coded solutions in the namespaces form the framework's class library and cover a large range of programming needs in areas including the user interface, data access, cryptography, web application development, numeric algorithms, and network communications. The functions of the class library are used by programmers who combine them with their own code to produce applications.

A C# application gets compiled in two steps. First, it is compiled into CIL (Common Intermediate Language) which defines the instructions for the CLR (Common Language Runtime). In the next stage, the code is compiled into machine language. Programs developed in different programming languages can be translated to CIL. Hence, .NET is programming language independent. C#, VB.NET or C++ developers can all be part of the same project without having to learn each others' languages.

3 The GAV Concept

GAV can be thought of as a collection of components that each performs a specific task. A component can, for example, read, filter or visualize data. By combining these components in a particular order, interactive GAV applications are created. The following components will be used during this laboration:

- LabExcelReader.ExcelReader
- Gav.Data.DataCube
- Gav.Data.ColorMap
- Gav.Data.LinearColorMapPart/Gav.Data.LinearHSVColorMapPart
- Gav.Graphics.ScatterPlot3D
- Gav.Graphics.ScatterPlot2D
- Gav.Graphics.Renderer

3.1 Data Flow

The code below shows how a data source is set as input to a filter and then the filter is set as input to the scatter plot.

```
filter.Input = dataSource;
scatterPlot.Input = filter;
```

3.2 Rendering

In GAV there are two main visualization components, the first one is called VizComponent, and the second one is the VizSubComponent. All visualization components in gav inherits from one of these. The purpose of this is that two or more components can be rendered on the same render surface. The first line of code below shows how a color legend is added as a child to the scatter plot. The scatter plot is a VizComponent and the color legend is a VizSubComponent. The next line show how the scatter plot is added to the Renderer. The Renderer class in GAV is in charge of when, where and how all components are rendered. Now, both the components will render on the same surface.

```
scatterPlot.AddSubComponent(colorLegend);
renderer.Add(scatterPlot, panel);
```

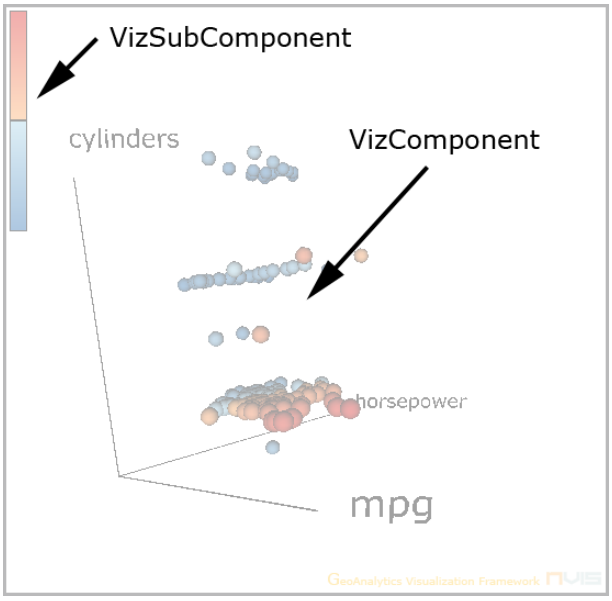

Figure 1: A color legend is a VizSubComponent and a scatter plot is a VizComponent.

3.3 GAV Documention

GAV has a documentation site that is currently under development.

Task 1:
Look at the documentation (API Reference) which can be found on the following address. *http://servus.itn.liu.se/courses/tnm048/GAVDoc/index.html*. Currently the documention only works under Internet Explorer.

4 Getting Started

The data set used in this exercise is a social science data set (SocialScience.xls). In a multivariate data set, a multivariate data item (or simply a data item) corresponds to a single row in the Excel spreadsheet and a variable (or dimension) corresponds to a column.

Task 2:
Open the Excel file and familiarize yourself with the data set.

Create a C# project:

1. Start Microsoft Visual C# 2005 Express Edition.
2. In the file menu choose New Project... In the resulting dialog choose Windows application and give the project a name.

3. After clicking the OK-button, save the project (Save All in File menu). The project must be saved in a local directory, for example D:. Don't forget to copy your project to your own (H:) directory before leaving the lab.

4. A project with a basic form is now created. The form is the graphical interface. Double-click on the file Form1.cs in the solution explorer to the right. The form is now shown in design mode. In the toolbox menu to the left (View->ToolBox), there are lists of graphical objects that can be added to the form later on.

5. In the toolbox menu, look under Containers. Drag and drop a SplitContainer control on the form. The SplitContainer's two panels will later on be used for the visualization components. Now is also a good opportunity to play around with C#. Add some controls (buttons, check boxes etc.) and give them some functionality to find out what C# and the .NET Framework is capable of.

6. Before you can use any of the GAV components you need to add a reference to the GAV Framework. Right-click on 'References' in the solution explorer and choose 'Add Reference'. In the resulting dialog select the Browse tab. Load the GAVLabFramework.dll.

7. Double click on the title bar of the form, this will generate a form load event handler and change your view to code mode (you can easily shift between design and code mode by right clicking the form in the solution explorer. The form load event occurs before a form is displayed for the first time and this is were you should initialize and setup all components and variables necessary for the visualization. For the moment though, you can leave it as it is. While in code mode you may notice the InitializeComponent method call made in the constructor. The InitializeComponent method is located in another file and witholds all the code generated in the design mode.

8. The data used in this lab exercise is in Excel format. A C# reader (LabExcelReader.dll) specifically written for this lab can be found in the same zip-file as this pdf. Add a reference to the dll by following the instructions described under step 6.

5 The Visualization

The main task in this exercise is to create an application that visualizes a multivariate data set in a 3D and 2D scatter plot. The data we will be using is the Swedish social science data. One multivariate data item maps directly to one Swedish municipality (kommun).

5.1 Import GAV namespaces

When you have come this far you should already have added all the necessary references to the project. However, to simplify the use of the GAV classes you should also add *using* statements for each namespace in GAV (*using* statements are placed in the beginning of Form1.cs). It is not needed, but allows you to use shorter names, instead of Gav.Graphics.ScatterPlot3D you can just write ScatterPlot3D. The following namespaces will be used during the lab.

- Gav.Data - Contains all components related to data manipulation.
- Gav.Graphics - Contains all components related to presentation of data.

- LabExcelReader - Contains the ExcelReader class.

5.2 Read Data

The first thing you need to do before you can visualize anything is to get the data from the Excel file into the program. *Tip:* Use the *out* keyword when passing the arrays.

Task 3:
Use the GetArrayFromExcel-method in ExcelReader-class. To do this, you first need to create one string array referance for the headers and a two-dimensional object array reference for the numbers. The two array references are passed as arguments to the method.

It is recommended to put all your data initialization code in a separate method, although it might seem unnecessary in this particular case it is good programming practise to do so.

5.3 Map Data

The data is now represented as a two-dimensional array in your program. In order to visualize the data with GAV, it must be represented as an appropriate GAV data structure.

Task 4:
Create a DataCube object and insert the data into it.

A DataCube is a wrapped three-dimensional array. During the lab we will only use two dimensions of this array.

5.4 Create DataCubeViewer

It is good practice to confirm that the correct values are inserted into the DataCube. GAV has a DataCubeViewer which shows the content of a cube.

Task 5:
Create a DataCubeViewer and set the created DataCube as input.

5.5 Render the Component

The component now exists but is not connected to a render surface. In GAV, the Renderer class handles all the rendering of components.

Task 6:
Create a Renderer and add the DataCubeViewer and a render target to it. The render target in this case is one of the two panels in the earlier created SplitContainer.

Tip: The DataCubeViewer needs to be invalidated before it can show the data.

Task 7:
Run the program by pressing F5. Compare the data in the DataCubeViewer component with the data in the Excel file. *Important:* You will get a LoaderLock error when trying to run the application. To avoid this, open Exceptions in the Debug menu. Under Managed Debbuging Assistants uncheck the LoaderLock - Thrown checkbox. This will fix the problem.

5.6 Create Scatter Component

The next step is to visualize the data items in relation to each other. There are several charts that can be used, but in this exercise we focus on scatter plots. Scatter plots are very useful since they give the user an easy and intuitive overview of the data.

Task 8:
Create a ScatterPlot2D-component and set the earlier created DataCube as input.

Task 9:
Add the scatter plot to the renderer. Don't forget to enable the component.

You should now see a scatter plot with two of the columns in the Excel data mapped to the axes. You can change which column to map to which axis by setting properties on the scatter plot.

5.7 Color - One Extra Dimension

The scatter plot is currently visualizing two dimensions of the data. It is now time to add another one, color. In GAV the coloring of glyphs, lines etc. is made by the *ColorMap*. The *ColorMap* is a collection of color map parts i.e. *LinearColorMapPart*.

Task 10:
Create a *ColorMap* and add at least one color map part to the *ColorMap*. Connect the *ColorMap* to the scatter plot. Try to change the *Index* property on the *ColorMap* (defult is 0). It sets which column to map.

The *ColorMap* will use the added color map parts to interpolate the colors between the max value and min value of the choosen column (*Index* property on *ColorMap*) in the data set. Each row in the datacube (same order as in the Excel sheet), will get a color assigned depending of its value in the choosen column.

Figure 2 shows the concept of ColorMap. The ColorMap in the figure is created by adding two color map parts, in this case a LinearHSVColorMapPart and a LinearColorMapPart (RGB). When adding two parts, one edge value at 0.5f is automatically created. The number of edge values are always the number of parts minus one. This means that if three parts were added, two edge values would be created at position 0.33f and 0.67f. The edge values can be changed by calling SetEdgeValue or SetEdgeValues methods on ColorMap.

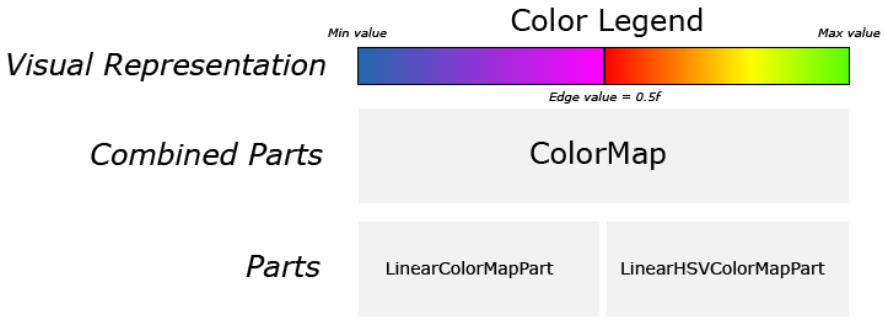

Figure 2: A ColorMap is built from ColorMapParts.

5.8 Brushing and Picking

Brushing and picking of data items are very important parts of infoviz, especially when working with linked views that show the same data. A user who can interact with the visualization components will be able to bring her knowledge and experience to bear on the problem. Without interaction, the user will be controlled by the visualization creator's view on how the data should be investigated. A user who is able to control how to watch the data will also feel more at ease and less frustrated, which also leads to better results.

Task 11:
Add picking functionality to the scatter plot. Use the *Picked* event handler and the *SetSelectedGlyphs* method on the scatter plot to enable brushing. The integer indexes in the pick list corresponds to the row numbers in the Excel file.

```
//Add the method Picked to listen to the Picked event.
scatterPlot.Picked += new EventHandler<IndexesPickedEventArgs>(Picked);

//This method is called if picking occurs in the scatterPlot.
void Picked(object sender, IndexesPickedEventArgs e) {
    // Do something with the picking information.
}
```

Tip: If you want another color for the brushing use the *SelectedGlyphColor* property on the scatter plot.

5.9 3D Scatter Plot

Task 12:
Create a ScatterPlot3D component and visualize it on the panel where the DataCubeViewier is rendered. Uncomment the code where the DataCubeViewer is added to the renderer.

Tip: Use the same ColorMap for both scatter plots.

Task 13:
Map the glyph size in the ScatterPlot3D component to a column in the data set.

Tip: Use the AxisIndexSize property.

5.10 Headers

Task 14:
Replace the x, y and z on the scatter plots with the names from the header array.

5.11 Linked Views

Task 15:
Link the brushing of glyphs so that if you pick in one of the scatter plots, brushing occurs in both components.