Spiegel - A Visualization Framework for Large and Small Scale Systems

Hans-Peter Bischof, Edward Dale, Tim Peterson
Rochester Institute of Technology
102 Lomb Memorial Dr., Rochester, NY 14623
hpb@cs.rit.edu, erd4819@cs.rit.edu, tjp4434@cs.rit.edu

Abstract. Visualization systems allow the visualization of n-dimensional data. In order to do so, the systems read the data, then convert the data into a phase space, and finally visualize the phase space. This paper discusses a visualization framework, which allows the creation of visualization systems ranging from visualizing the behavior of atoms under the influence of temperature to colliding galaxies and the spread of the plague in 1347 A.D. in Europe. The visualization framework which we present is extensible, which means a user can easily add new components to the framework and create visualization systems that could not be anticipated at present time. The paper will give an overview of the framework, and will be focusing on the graphical programming environment and the extensibility of it. The last part of the paper will focus on how a simulation system can interact with the visualization, and for example inform the visualization system of interesting events.

Key words: Visualization framework, pipeline architecture, dense stellar systems

1 Introduction

Visualization systems play an important role in interpreting vast amounts of data. The simulation run of a dense stellar system [2] creates simulation data of a size greater than 20 GB. It is virtually impossible to analyze the dynamic nature of the simulation without having a visualization of the data. Very often the data is transformed into a different phase space first, which is then visualized. For example it can be of interest to visualize the mass density distribution [3] of colliding galaxies and not the positions of the individual particles.

In this paper we will discuss Spiegel. Spiegel is a visualization framework that supports the creation of specialized visualization systems by combining existing or new tools.

A visualization system is a tool which allows one to visualize data. A visualization framework provides facilities to program its components in order to create a visualization system. It should be easily extensible to meet future visualization needs that are not currently supported.

Spiegel is a visualization framework for both large and small scale systems. Figure 1 shows an example to two different visualizations. The left image shows the protein 1D66. The backbone of the protein, which is the connection made up of a chain of carbon atoms called alpha-carbons, is highlighted. The right image shows the collision of three galaxies. The big red balls are black holes, the particles of each galaxy is shown as single pixels in different colors.

In order make a visualization, the following is needed:

– Visible attributes of an object. The visible attribute of an protein could be spheres with a color indicating the kind of atom it is. The spheres would be arranged at the corresponding x, y, z positions in space. Artificial visible attributes must be created for immaterial attributes like density.
– Cameras which can look at the scene, which is constructed of the visible attributes, like the spheres of a protein. The camera can view any scene that has been created, and therefore it is independent of any specific visualization scenario.
2 Visualization System versus Visualization Framework

There are two general approaches to implementing a visualization system: A very focused and specialized visualization system, or use of a very general visualization framework.

The general functionality of a visualization system is described in Figure 2. The data is read from an input stream, converted into a internal data structure and filtered based on the request of what should be visualized. The extracted data is then sent to a module which converts the internal data structure to a visual representation which will be shown on the screen. User input allows one to change the behavior of the visualization as far as the visualization system allows.

A focused and specialized visualization system is designed to perform a specific set of tasks. For example, Xnbody [6] is a very nice tool to visualize a specific set of attributes of stellar simulations, like orbits of particles. Xnbody becomes useless if the user is interested in the visualization of attributes which Xnbody was not designed for, like the density of the stellar space using the algorithm described in [3]. Even very minor modifications are extremely difficult in Xnbody [7]. The same difficulties are faced by the web interface [4] to NEMO and a VRML approach to visualizing astronomy data [1].

Spiegel is an environment which allows one to create visualization systems. Spiegel provides the functionality to add new components to the existing set of components, allows for connections between the components, and provides a communication channel between the components.
3 Spiegel

The Spiegel visualization framework is an environment, or a toolbox, which facilitates creating visualization systems. A visualization system in Spiegel terms is nothing more or less than a Spiegel program. Spiegel is written in Java and utilizes Java3D for visual effects.

A component in Spiegel is equivalent to a Unix™command. It can have arguments, like a Unix™command, but it can have multiple typed in- and output streams. The Spiegel programming environment is equivalent to a Unix™shell which allows to write a shell script. The Spiegel runtime environment is equivalent to the Unix™OS in which the shell script is executed.

Spiegel is a flexible, programmable visualization framework which uses the Unix™pipeline concept for data flow. A plugin technology is applied to make the system extensible. Spiegel also includes flight path and movie making capabilities.

Spiegel’s pipeline structure assigns a data type to each input and output stream of each component. Only matching communication points can be connected. Additionally, an output stream can connect to multiple input streams and an input stream can connect to multiple output streams, if the respective components allow it. Each component can be fine-tuned by using component-specific arguments.

The main functionality of Spiegel is defining communication channels between the components of the visualization system and transmitting the data from the sending component to the receiving component. Spiegel ensures that output streams can only be connected with matching input streams.

Spiegel has a graphical programming environment in order to create visualization systems. Using this environment is one way to create Spiegel programs. The programs are stored as text files, so a standard text editor can also be used to create a Spiegel program. A Spiegel program is shown in its graphical representation in Figure 3.

In Spiegel pipelines the data flows from left to right. The file component is an argument for the star extractor. The star extractor opens the file specified in the file component, reads the data, and converts the data into 3-dimensional star objects and sends the star objects to the two star cameras. One star camera looks at the scene from the top, and one camera looks from the side. Figure 4 shows what would be seen on a computer screen. The constant argument defines the size of the black holes in the visualization. The clock moves the visualization through time. At each clock tick the star extractor extracts the data for the current simulation time.

Each Spiegel program has a textual representation. When the system starts up, the program is read from a file. The visualization system then reads additional statements from standard input.
4 Control

Data flows through one communication channel, starting from the simulation, moving through the components and ending up in a display. A second communication channel provides control of the components of the visualization system. Each component defines the set of commands it understands. This means that the set of statements which can be issued to control the visualization system is dependent on the components that are in use. The parser, which evaluates the statements, is configured dynamically during runtime of the visualization system.

5 Graphical Programming Environment

The graphical programming environment (GPE) is the part of Spiegel which is used to create visualization systems. The GPE allows one to select components and connect matching input and output communication endpoints. After a communication endpoint is selected, the GUI highlights all of the components which have a matching endpoint. The GPE also allows configuration of component-specific parameters, such as the position of a camera, the size of a black hole, etc. The visual representations of connection points are small lines sticking out of the component. The name of the connection is shown on the screen when the mouse is positioned over the line. A video of how to use the GPE can be found on the web [5].

The constructed program can be stored in a file. Every useful programming environment must be capable of creating abstractions, i.e. creating functions and using them. Spiegel’s GPE allows the creation of functions which can be reused later on. Figure 5 shows the function \texttt{star camera}.

It is clear that the developer of a new component is not interested in developing the graphical user interface aspects of the component. The framework takes care of this. A developer needs only specify the name and type of the component’s inputs and outputs. At runtime the framework inspects all existing components and creates the graphical representation of the component as needed. Figure 6 shows a snippet of the code for the \texttt{Camera3D} component. Lines 2 and 3 declare an input parameter named \texttt{location} of type \texttt{Point3d}. This code will create a small line on the GUI which is marked \texttt{location}. Lines 7 through 10 define a statement which will be understood by this component: \texttt{set location x y z}. Each occurrence of \texttt{text(3) slider(-10, 10, 0.1)} defines a GUI component which will allow assignment of a value to that field via the GUI. Figure 7 shows what the resulting part of the GUI looks like.
public static DataInput.Info[] inputInfo = {
    new ParamInput.Info<Point3d>(
        "location", new Point3d(5, 5, 5), Point3d.class),
    ... 
};

public static String[] commands = {
    "set location " +
    "<double x text(3) slider(-10, 10, 0.1)> " +
    "<double y text(3) slider(-10, 10, 0.1)> " +
    "<double z text(3) slider(-10, 10, 0.1)>",
    ... 
};
6 Extensibility

What must be done in order to develop a new component? Suppose a component should be added which visualizes the density of a dense stellar system. There is already a component which can read a file and can convert the information into star objects (see Figure 3). What is left to do, is read the star object and calculate the density. Needed is a new component which has an input connection of type star objects and an output which can be understood by a 3D camera, such as CamTop in Figure 8. The following code snippet sketches out what needs to be done. Lines 2 through 5 declare the input and output communication channels. The update() method will be called whenever the system requires the output data to be updated. Line 8 creates a BranchGroup (a type defined by Java3D) which will be filled with the result of the densityCalculation(). Line 9 retrieves the star object information from the input connection, and line 11 sends the computed data to the output connection. The most difficult part of this component to write is the content of densityCalculation(), and the framework does not provide any assistance for this method.

```java
public static DataInput.Info inputInfo =
    new SingleDataInput.Info<StarIDMap>("starsIn", StarIDMap.class);
public static DataOutput.Info outputInfo =
    new DataOutput.Info("streamOut", BranchGroup.class);

public void update() {
    BranchGroup densityVisualization = new BranchGroup();
    theStars = starsIn.get();
    densityVisualization.add( densityCalculation(theStars) ) ;
    streamOut.set(theStars);
}
```

Fig. 8. A Density Component

7 Finding the Needle in the Haystack

The visualization system has no knowledge about the physics of the simulation. Therefore the visualization system cannot automatically direct the camera to the events which are of interest to scientists. On the other hand, the simulation system has full understanding of the physics and knowledge of when an interesting event occurs. Each component of Spiegel can be controlled via statements send to the Spiegel system. Therefore is is possible to change the behavior of each component, like

− move a camera to a given position
− create a new camera
− change the appearance of the visualized objects
− etc.

from the simulation system.

This will allow the simulation system to control the behavior of the visualization system.
8 Conclusion

We presented in this paper a visualization framework which is extensible and can be used to visualize $n$-dimensional data ranging from small scale to large scale environments. The component and pipeline approach makes this framework extremely flexible. The decision to create a control language enables easy control of the system.

Acknowledgements

We would also like to express our thanks to David Merritt, and Stefan Harfst from the Department of Physics at RIT. They provided the actual data and the interpretation of the visual representation. Additionally we would like to thank Piet Hut from the Interdisciplinary Studies Institute for Advanced Study at Princeton for wonderful discussions. Finally we would like to thank the Computer Science students at RIT who helped to build this system.

References