How Data can become Data a Movie Star?

H.-P. Bischof
Department of Computer Science,
Center for Computational Relativity and Gravitation,
Rochester Institute of Technology, Rochester, NY, US

Abstract— *Visualization of scientific data is the art of converting numerical values into an image, which can be comprehended by a human visual system. The conversion process can lead to a movie or an individual image. The creation parameters, like light, viewing position, etc. typically do not change. A movie can be assembled out of images where the creation parameters for each individual image does not change, or each image may have different creation parameter set. This article discusses the second approach, which is a significantly more dynamic and creative approach. We will analyze what existing visualization systems are capable of, and we will present a unique way how to create accurate, but very dynamic movies.*

Keywords: Visualization Framework, Animation

1. Introduction

This conversion of numerical data into visuals makes it sometimes possible to understand complex relations, but it is no replacement for an analytical analysis. A histogram, for example, is the appropriate to tool to convey statistical information to statistically untrained people[10]. A visualization of the functions $f(x) = e^{-1}$ and $g(x) = 1x$ is helpful in order to see that $f(x)$ and $g(x)$ for x close to 0 are nearly identical. The proof for the birthday paradox[11] becomes a bit easier if this fact is know.

The visualization of scientific data should not be confused with the generation of a visual for a movie. The outer space scene generated for the movie WALL-E[5] is wonderful, but not based on data generated by an experiment or a simulation. A visualization of scientific data needs to be true to the science, but can also be exiting from an experience point of view. In order to do so, it must be possible to control viewpoint, color, light, size of objects and other attributes used to control the creation of the visualization.

For a moment picture the following visualization problem. A tiny black hole is merging with a very large black hole, and for the scientist it is important to show the correct size of the objects. In order to see the path of the small black, the viewpoint has to be so far away that the small black hole becomes invisible, because its appearance is smaller than one pixel from the desired viewpoint. One solution of this problem would be something like this: The viewpoint moves from a viewing point far away to the original sized tiny black hole. The at first invisible black hole will became visible during the movement of the viewpoint. After the final viewpoint is reached, the size of the tiny black hole is increased to a size, which makes it visible form the original viewpoint. The viewpoint then moves to the desired viewpoint. The merger of the black holes continues and shortly before the merger takes place we use a similar technique as described before to show the actual merger. The viewpoint moves close to where the merger takes place, then the size of the tiny black hole shrinks to its real size and the merger continues from there on. It is obvious that the movement of the viewpoint have to be smooth in order to create a pleasant viewing experience. For example, the movement cannot come to a abrupt stop at the end of a movement.

The rest of the paper explores three selected visualization systems in terms of their functionality and their capability to create more animated visualizations. The systems chosen are visit[1], ParaView[2] and Spiegel[7].

2. Architecture

All three visualization systems are using similar data flow architecture. The raw data is read, and filtered if the raw data cannot directly be used for the visualization. This data is then converted into geometric objects. These objects will then be visualized according to the chosen algorithms and then stored, shown, or assembled to a video.

ParaView is normally controlled via GUI, but the engine can be programmed using the scripting languages pvbatch[3] or pvpyhton[4]. The batch programs allow creating, connecting, and controlling the data flow and the components used in the process. The flavor of scripting language is similar to other scripting languages. The use of it requires a significant understand of the available ParaView components and their communications and control. The scripting languages are the path to the creation animated movies. This aspect will be explored in chapter Architecture.

The Spiegel visualization framework is a Unix pipeline inspired architecture. Similar to ParaView a GUI is used to create the data flow. A scripting langue used to create and connect the components, and the resulting program is then stored in a file. The scripting language is extremely simple and does not have control structure, or variables differently to the scripting languages used for VisIt or ParaView. The creation of more dynamic movies is controlled by a Spiegel

component which if programmable. This aspect will be explored in chapter Animation Language.

VisIts dataflow is based on operators, which are performed on the data, like slicing, etc., before the modified data actually gets visualized. Like the other two systems, a GUI controls visit. VisIt is using a scripting language or a GUI component in order to create animated video. The GUI component allows very simplistic animations; the language needs to get used in order to create sophisticated animated movies.

3. Visualization Components

A Spiegel visualization component[7] has typed input, output and argument channels; k input channels, l output channels and m argument input channels. The output of a component can be connected to the argument of a component, which allows reaching into the data stream and modifying the visualization based on the data. Lets assume we would like to follow a black hole with a cameras viewfinder. In order to do so one component of the visualization program would reach in the data stream and send out the position of the black hole we are looking for. This position would be sent as an argument input to the camera, and the viewpoint would follow the black hole. The camera itself would be at a fixed position in space, but the cameras viewfinder, or look-at-position would follow a particular black hole. This functionality alone would not do, because the black hole may ay be in a less than desirable lighting situation. One solution would be to point a light source to the same position as the look-at-position of the camera. Figure 1 illustrates this concept. The dashed lines connect output channels with a argument channel, and the solid lines connect output with input channels. The Stars component send out all stars to the Visualizer and the component, which extracts the position of the black hole, we are interested in, in this case number 3.

This “hello world” program illustrates one design principle behind the Spiegel system. The Spiegel program language is simple; it allows only creating and connecting components. Every functionality else need is implemented in Java, which allows to use object oriented program paradigms and the power of existing Java frameworks. This concept allows to add functionality very easily, because the Spiegel specific communication part can be learned in less than 5 minutes. My students have proven this many times.

VisIt and ParaView are using well known scripting languages, mainly python. The connection of data output and argument does not really exist. The burden of the functionality of the visualization program is divided into scripting and the components. This means, the same functionality can be achieved in many different ways. Adding new components into the ParaView or VisIt environment is doable, but very difficult. My students have proven this many times. In other words, an animation is most likely done in a scripting

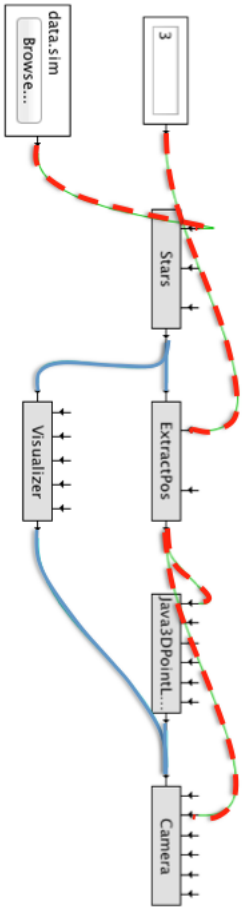

Fig. 1: How to change the viewpoint based on the location of a black hole

environment, and it is not easy to re-use established ideas, from one visualization to an other.

4. Visualization Algorithms

The basis for visualizations is obviously the available algorithms. VisIt and ParaView trump Spiegel in this regard significantly. Benger at all[6] describe in detail the usual implemented visualization algorithms. VisIt and ParaView implement a super set of these algorithms. Both systems have been developed to be a production tool. Spiegel, on the other hand, was and is developed as research system and its main purpose is to understand design criteria’s for visualization systems. Spiegel has been successfully used to create visuals as shown on History Channel.

5. Animation 1.0

The animation of a scientific data visualizations can enhance the understanding of the data[8]. The result of an animation is a movie; therefore we must keep in mind how many frames per second the animation creates. In other

words, the number or frames used for the simulation is know before the animation is created.

The most simplistic animation of a supernova simulation of a stellar explosion is keeping every thing constant, viewing position, lighting condition etc. The input parameter for this animation would be t, meaning we create a frame for t starting at the beginning, and then the next value of t would be calculated by t = t + Δ t. This might not the most desirable animation because at the beginning at the end of the simulation nothing of interested may happen. In other words, it is known when and where the visualization shows an interesting situation. Therefore it would be better if Δ t would not be a constant, but rather be calculated as Δ t = F(simulation time) in order to fast forward the time of show the simulation in slow motion. If this is done, then we introduce the concept of a simulation time and visualization time. The simulation time is the time of the simulation, which created the data; the visualization time is the wall time passing while the animated movie is running. Scalars like time are very obvious and easy to understand and to use. The visualization time moves forward in constant units; the simulation time is calculated based on a function. At this point the how to calculate the simulation time is not important. We will be discussed in the next chapter.

Is a scalar, like time, the only kind of data which drives an animation. For example, lets assume you would like to visualize how a projectile rips a balloon apart. It might be preferable to use the x/y/z position of the projectile to drive the simulation and not the time. This is most likely not the path to go, because if the time is know, then it is know there the projectile is, because the projectile has to follow the laws of physics. The author has never found a need to use to anything else than a scalar value to control the animation. Nevertheless the following chapter will not use this restriction.

6. Animation 2.0

Animation always require at least on input parameter, i, to drive the animation, which at the end controls the change of the attributes used to create single frames. We assume at this point, that the input parameter is a scalar value. A function is needed for every attribute controlled by i.

For this example we would to adjust the viewpoint of a camera. The viewpoint should follow a path in 3d. We assume i moves forward by a constant Δ i. We will later show that this is a fair assumption. The path p, is defined by a function $f(i)$. We need to know the values of $f(i)$ for a few fixed values of i. These values are called the anchor points. For example:

```
I = 0: p = (0, 0, 255)            // a
I = 1: p = (0, 255, 0)            // b
I = 3: p = (255, 0, 0)            // c
I = 4: p = (255, 255, 255)        // d
```

The question is how do we calculate the value for p(i), at i = 2? Some kind of interpolation has to happen between the anchor points. If the values represent the RGB values for a color, then a linear interpolation between the points is the one to choose. TCP-splines[9] would be the right choice, if the values represents the path of a flying seagull, because the flight seagull does not include sharp kinks.

The Δ for i is: 1, 2, 1. Lets follow the flight of the seagull for a moment. We note that the speed of the seagull between point a and b is twice as fast as the flight from point c and b. This technique would be used to create a speed up or slow-motion view of the simulation.

It is obvious that the described algorithm can be implement in any reasonable language, which includes the scripting language used in ParaView or VisIT. The question is would be worthwhile to implement a tiny specialized language animation language.

7. Animation Language

The input for an animation language is one input value. The outputs can be many attributes as a function of the input. The values of the attribute are controlled by the input value, the anchor points, and the interpolations used for a given attribute. We have to keep in mind, that the final result will be a movie; therefore we have to define how many frames per second the animation should generate: The animation language must allow for the following:

1) Define the outputs attribute by names, we call this objects output streams
2) Define the anchor points for each output stream
3) Define the interpolation algorithm for each output streams

An example is shown in Fig xxx and the use case is show in Fig .. The keywords of the language are: fps, stream, name, type, interpolator, and the parenthesis. TCB is an index in a table defining the one used for the position stream.

```
fps 25.0

stream position {
        name "position"
        type vector interpolator TCB
}
time 0 { position(0, 0, 255) }
time 1 { position(0, 255, 0) }
time 3 { position(255, 0, 0)
time 4 { position(255, 0, 0) }
```

The component Flythrough gets the input parameter from the Clock component and sends out the position parameter calculated based on the animation program shown in Figure 2. The clock ticks forward in time, and this will trigger a change of the output attribute position.

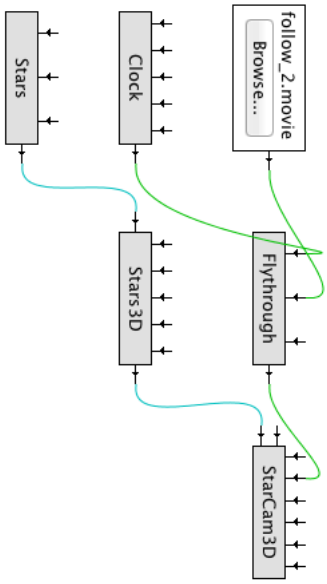

Fig. 2: How to change the viewpoint based on the location of a black hole

8. Extension

The component parsing the animation programs needs to get modified, if the grammar of the animation language needs to change. This might be a serious undertaking, highly depending on the new grammar. Adding new interpolators, or new type is trivial; it just requires the creation of a interpolator class and an extension of an array to connect the identifier with the interpolator.

9. Future Work

Incorporating of the Spiegel animation philosophy in ParaView or VisIt should be done as a proof concept. The animation language does not make it easy to modify the program after it is written because the language does not have variables and arithmetic build in. The language would benefit from adding these two ideas. Rather complicated animated movies have been produced with this technology, but none of them had audio attached to it. It would be interesting to understand if this language could support the creation of audio for the animation.

References

[1] The visit home page @ONLINE. March 2012.
[2] Paraview - open source scientific visualization tool@ONLINE. March 2013.
[3] Paraview/users guide/batch processing @ONLINE. March 2013.
[4] Praview/python scripting paraview @ONLINE. March 2013.
[5] Wall-e@ONLINE. March 2013.
[6] Werner Benger, Markus Haider, Josef Stoeckl, Biagio Cosenza, Marcel Ritter, Dominik Steinhauser, and Harald Hoeller. *Visualization Methods for Numerical Astrophysics*. InTech - Open Access Publisher, 2012.
[7] Hans-Peter Bischof, Edward Dale, and Tim Peterson. Spiegel - a visualization framework for large and small scale systems. In *In MSV âĂŹ06: Proceedings of the 2006 International Conference of Modeling Simulation and Visualization Methods*, 2006.
[8] Danyel Fisher. Animation for visualization: Opportunities and drawbacks @ONLINE. March 2013.
[9] Doris H. U. Kochanek and Richard H. Bartels. Interpolating splines with local tension, continuity, and bias control. In *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pages 33–41, New York, NY, USA, 1984. ACM Press.
[10] Douglas C. Montgomery. *Introduction to statistical quality control*. Wiley, New York, NY [u.a.], 3. ed edition, 1997.
[11] Kazuhiro Suzuki, Dongvu Tonien, Kaoru Kurosawa, and Koji Toyota. Birthday paradox for multi-collisions. *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, E91-A(1):39–45, January 2008.

10. Conclusion

The use of this or any other animation language could be added to ParaView or VisIt. Animation in these two environments is difficult to create. Therefore it would be beneficial for the users to add visualization language as described