Using CavePainting to Create Scientific Visualizations

David B. Karelitz Daniel F. Keefe David H. Laidlaw

Brown University[*]

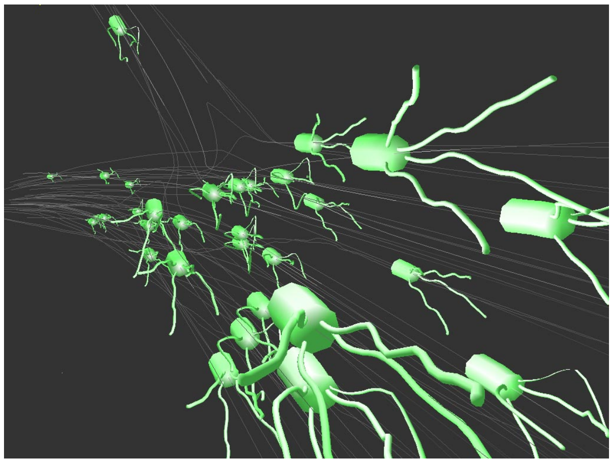

Figure 1: We extended CavePainting, a system for drawing in VR, to aid design tasks in the scientific visualization domain by allowing designers to easily preview designs in an immersive environment. This figure contains one prototype of a particle designed to show pressure as the width of the head, and velocity as the position of the tentacles with faster particles having more streamlined tentacles. The legend used to generate this image is shown in Figure 2

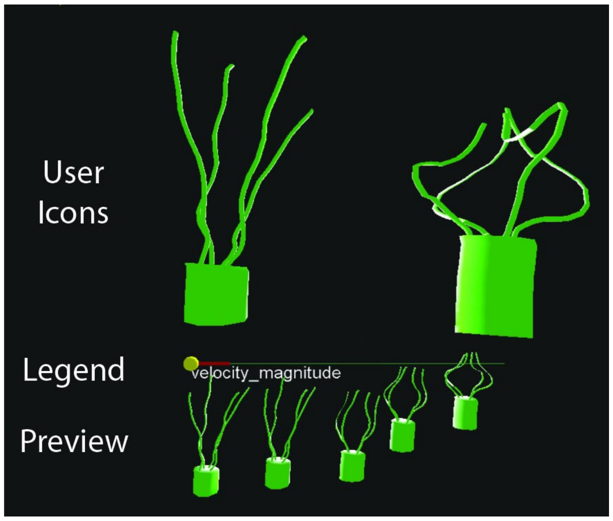

Figure 2: Legends are used to link drawn icons to data. Velocity Magnitude, the datatype represented by this legend is shown just below the green line. User drawn icons are added above the line, and a preview of the final icons or particles is shown below it.

Abstract

We present an application of a virtual reality (VR) tool to the problem of creating scientific visualizations in VR. Our tool allows a designer to prototype a visualization prior to implementation. The system evolved from CavePainting [Keefe et al. 2001], which allows artists to draw in VR. We introduce the concept of using an interactive legend to link a visualization design to the visualization data. As opposed to existing methods of visualization design, our method enables the researcher to quickly experiment with multiple visualization designs - without having to code each one. We applied this system to the visualization of coronary artery flow data.

1 Introduction

According to Senay and Ingnatius, “The primary objective in data visualization is to gain insight into an information space by mapping data onto graphical primitives” [Senay and Ignatius 1994]. The first step in this process is often a quick sketch of the elements of the visualization. When designing visualizations for VR, sketching on paper does not capture the immersive nature of VR. Furthermore, implementing each design often takes hours or even days as visualization styles are coded, examined, and evaluated. The goal of our system is to reduce the iteration time for designing a visualization to a few minutes. We accomplish this by allowing an artist to sketch a visualization in VR, and then apply that to the actual visualization data. The end result is a hastened research cycle; each design can be implemented and evaluated in a matter of minutes.

[*]Department of Computer Science, Brown University, Providence, RI 02912, {dbk,dfk,dhl}@cs.brown.edu

The CavePainting system allows users to directly draw 3D forms in virtual reality using a six degree-of-freedom tracker. The user manipulates a brush to generate a stroke of color and texture. These strokes can be edited and combined into compound strokes.

1.1 Motivation

The existing artery application visualizes complex fluid flow using particles. The particles showed only the path the particle would take through the flow; however, simply looking at the path of a particle was not enough to give a comprehensive image of the flow. The flow is characterized by multiple values at each point within the flow. The main problem then became how to show multiple values in a single particle. [Sobel et al. 2002]

The traditional method of designing particles is to sketch some designs on paper, implement them, and then evaluate them. The main problem with designing VR images on paper is that paper design does not fully characterize what the resulting visualization will look like in a VR environment. For example, choosing colors for VR is difficult to do on paper since the projected colors are often dim and unsaturated. Furthermore, any design on paper is still a 2D design, and 3D designs on a 2D medium may have problems when viewed in immersive 3D. Our system operates between the paper design and the actual implementation, and provides a medium in which to easily test a design. Paper designs are still useful as a starting point, but refining a base design can proceed much faster with our system than with the design and implementation cycle normally employed. Using our system, a researcher can take a paper design, sketch the design in 3D, and view the final result.

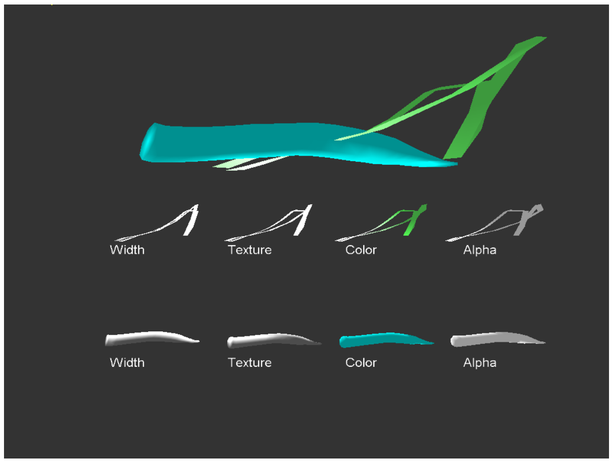

Figure 3: A CavePainting stroke comprises a path and a style. There are two paths and two styles in this example, the blue base and the green wing. Parts of the style, such as width, texture, color, and alpha, as depicted above, can be selected and modified using this view.

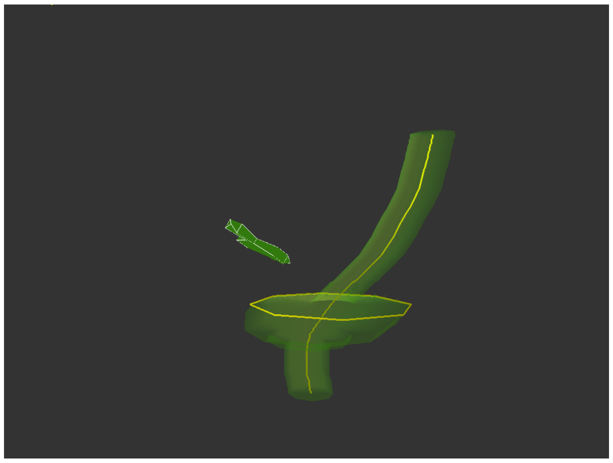

Figure 4: The width of a stroke is first modified by selecting the width attribute. This causes the stroke to become transparent, and the path, the yellow line along the length of the stroke, to be drawn. The new width of the stroke at the yellow ring is the distance between the path and the paintbrush.

1.2 Our Approach

Legends were used to combine CavePainting strokes with the data being visualized. CavePainting strokes added to the legend indicate how the final visualization element changes in response to a particular data type. The lower portion of the legend shows miniature versions of the final visualization element. There is one legend per data attribute visualized.

The previous system used for visualizing artery flow data requires each different visual style to be explicitly coded, and as a result, adding new styles or more information to the visualization often takes days or weeks. With our system, the researcher is able to sketch the legend for a visualization and see the result almost instantly.

2 Methods

The system is based on three critical components. The first component, the stroke, is the drawing primitive for generated icons. The second component is the legend, which links icons to actual data. The third component is the interpolated particles generated by the system.

2.1 CavePainting Strokes

Again, the basic element in our system is the stroke. A stroke consists of two main parts, the path of the stroke, which is defined by the path of a 3D tracker through space, and the style of a stroke. The stroke style consists of a form and style properties. A stroke's form is a cross section of the final stroke. In our program, we support a flat cross section, which results in a ribbon, and a circular cross section, which results in a tube. Style properties are contained in forms, and are organized into layers, with each layer having texture, color, and transparency. Color and transparency are defined along the length of a stroke, so different parts of a stroke can have different colors. Strokes can also be combined to form compound strokes or icons. In this case the system stores an offset path in each form, and applies that offset to the path of the stroke to obtain the final shape and position of that part of the compound stroke. In Figure 3 two forms are visible, the straight base, and the curved wing. Each form has width, texture, color, and alpha styles, though texture and alpha are at their default values of no texture and no transparency respectively.

Strokes are edited by first selecting the property to be modified using the exploded view of a stroke as seen in Figure 3. There are two types of interaction when modifying a stroke. When modifying the width of a stroke, the stroke becomes transparent and the path is shown. Then the system chooses the point closest to the brush, and makes the width of the stroke at that point the distance between the path and the brush, as seen in Figure 4. Color is modified by shooting a ray out of the brush and modifying the closest point on the path to that ray, as seen in Figure 5.

2.2 Data

The data we are using approximates a bifurcated coronary artery using a large slightly curved arterial pipe with a smaller pipe joined at an angle. The data is the result of a numerical simulation using this model. The complete dataset is time varying and simulates a single pulse of fluid from the heart. It contains multiple datatypes at each point, such as pressure, velocity, and vorticity.

2.3 Linking Strokes to Data

Legends were chosen to link example icons to the data being visualized. There is one legend for each datatype in the dataset. The legend, as seen in Figure 7, consists of three parts. The first part is the icon for the legend which contains the data type associated with the legend and the direction of increasing data value. In this case pressure is increasing to the right. The user draws example icons above the legend, e.g. the two large birds. Once an icon is added to the legend, a preview of example particles appears, e.g. the smaller birds below the line. The preview shows how the icon changes for the datatype of the legend, and assumes all other datatypes are at half value.

In order to generate a final particle, there are three steps. First, each legend examines the example icons it contains to determine which attributes of the icon should change in response to this data type. Second, each legend generates a sparse placeholder icon that contains only those attributes that change in response to its particular data type. Finally, the sparse placeholder icons are combined with an initial icon to generate the final icon. In cases where more

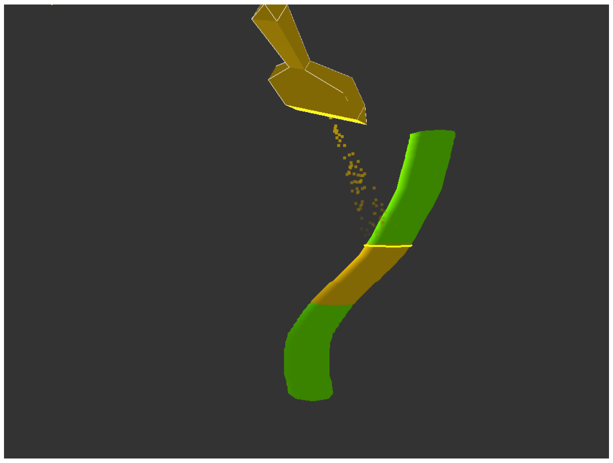

Figure 5: The color of a stroke is modified by selecting the color attribute of a stroke, then spraying the stroke with the new color. The yellow ring indicates the current position on the stroke. The same interaction is used for editing the transparency of a stroke, with white being opaque and black transparent.

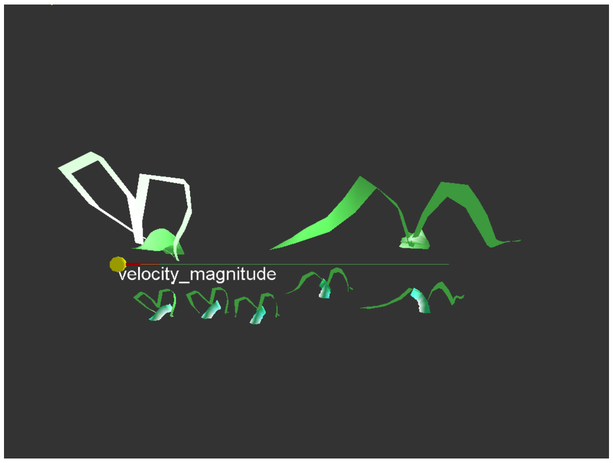

Figure 6: This legend was used to map speed to wingspan. The user-drawn icons appear above the line; below it are some samples of the final particles.

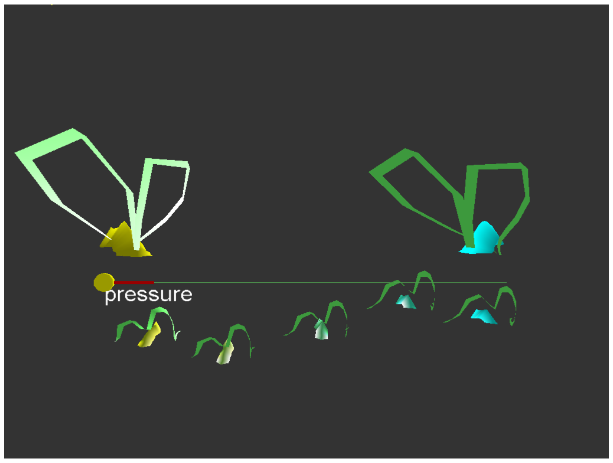

Figure 7: This legend maps the color of the birds to pressure. Again, the user-drawn strokes are above the line, and the final icons are below.

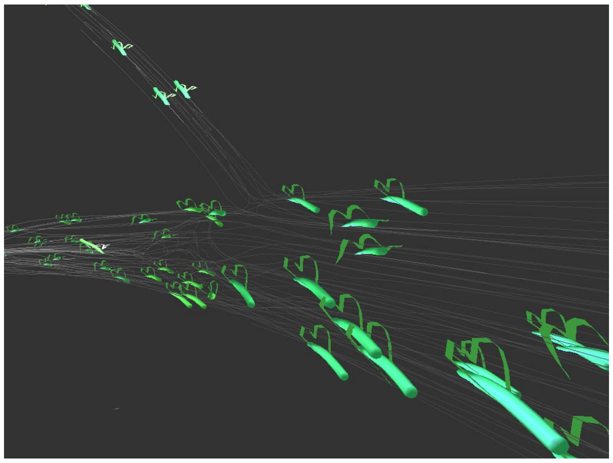

Figure 8: This example visualization shows bird icons that change wingspan in response to velocity, and color in response to pressure. This snapshot was taken at a low pressure point.

than one datatype submits a change to the final icon, the changes are interpolated. In the bird example, The pressure legend generates sparse strokes containing only the color of the bird base. The velocity legend generates a sparse stroke containing a path for the bird wing. These two elements are merged with a copy of one of the example strokes to form the final particle.

3 Results

We used the system to design particles for the visualization of the artery data. The CavePainting system excels at creating organic forms, so we chose some organic creatures – fish and birds – as a basis for the particle design. Both particles were created to simultaneously show two data types: velocity and pressure. Overall, it took about half an hour to generate each visualization.

The first particles were squid with some trailing tentacles, as seen in Figure 10. Speed was mapped to the shape of the tentacles; contracted tentacles signify a slower velocity and streamlined tentacles signify a faster velocity; pressure was mapped to the size of the squid's head.

The second set of particles created were modeled after birds, as seen in Figures 8 and 9. The birds show velocity as the shape of the wings, outstretched for fast, and folded in for slow; they show pressure as color, with red as high pressure and blue as low pressure. The legends used to create bird icons are also shown in Figures 6 and 7.

Some other particle designs were tried, such as leaves and raindrops. There were two main problems with these designs. First, the icons were much too complex, and only a few could be shown at reasonable frame rates. The second problem was that the example strokes were not sufficiently different, hence it was near impossible to determine the actual data values represented by the icon.

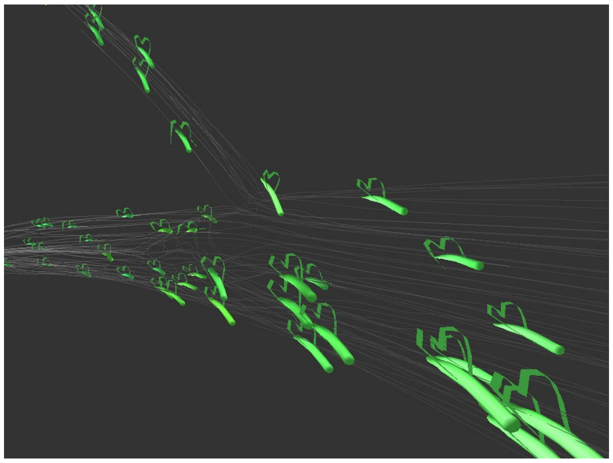

Figure 9: This is another snapshot of the bird icons, this time at a high pressure point.

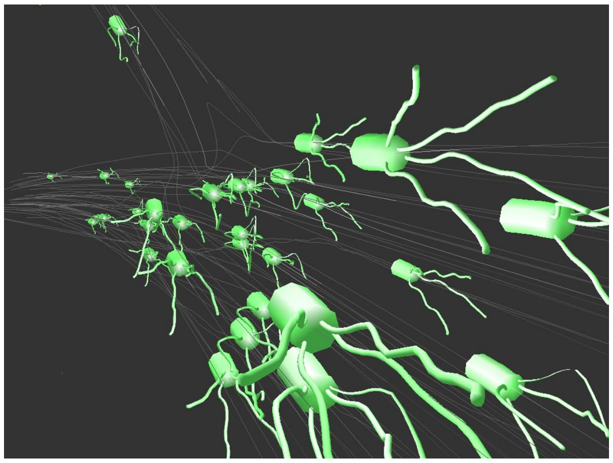

Figure 10: This icon shows squids moving through the flow. Velocity is mapped to the shape of the tentacles - slow yields compacted tentacles and fast yields streamlined tentacled. Pressure is mapped to the width of the squid's head.

4 Discussion

4.1 Evaluation of the particles

We postulated that the squid particles would represent velocity well because the particles exhibit movement well. There was a major unanticipated problem with the squid's initial design. When many squid were in close proximity, the tentacles overlapped and it became extremely difficult to determine which tentacles belonged to which squid. This problem was alleviated with dynamic lighting, though that had the consequence of altering the colors. Since color was not used in this visualization, this was an acceptable solution. Another viable solution would have been to either apply a texture to the tentacles or to change the color in the middle of the tentacles.

The second problem with the squids was that it was difficult to determine the relative size of a single squid's head, and in turn the relative pressure at a single point. It was still possible to see general pressure trends, but it was near impossible to determine a pressure value at a single point. Overall, the squids were well suited to visualizing velocity, but not pressure. Thus the visualization was not an improvement over the initial particles.

On the other hand, the birds turned out much better than we anticipated. It was very easy to tell the position of the wings by looking at the particle, and since the birds were not very long, the problem of intersecting particles was minimized. Furthermore, the color change in response to pressure was extremely easy to see, and the pressure value could be inferred from examining a single bird. Compared to the squid, the birds were able to show two values simultaneously using a single particle.

4.2 Speed and Flexibility

The relationship between speed and flexibility in our system was one of our paramount problems. The goal was to allow enough flexibility in particle design, yet still be able to generate enough particles so their interaction with each other could be understood. In informal testing, our system was able to generate about twenty-five percent as many particles as a hard-coded implementation of a similar particle design. While this was fewer than we had hoped for, it was still enough to allow testing of future particle designs. There is definitely room for improvement in this area, with one possible approach being to offload the interpolation of the particles to the graphics card.

4.3 Problems with Implicit Determination of Interpolation Variables

The main problem with this system was that it implicitly tried to determine what properties of a stroke to changed for each data type. This was necessary since particles would tend to gravitate towards an average particle if all the particle properties from each data type were averaged to generate the final particle. Implicitly determining what to interpolate for each data type was problematic because the user invested more thought in the particle design than in actually drawing the picture. One possible solution would be to have the user draw the particle, then explicitly choose what parts of the particle to interpolate for each data type. Although both result in correct interpolation, it is easier and takes less time to explicitly choose which attributes of a stroke to interpolate than to draw the icon correctly.

However, a much more difficult challenge exists: ensuring that particles have the same stroke structure and stroke direction. Unfortunately, this imposes limitations on the design process of a particle. Mainly, the designer must be aware of the implementation of the system in order to use it effectively. The simplest solution is to try and correlate the endpoints of the stroke, or the initial direction of the endpoints of the stroke. Interpolating particles with different stroke structure can be accomplished by splitting and merging strokes. When and how to perform these actions has yet to be explored.

The consistent theme among all these problems is how much the user should explicitly instruct the system versus how much the system can infer from the users actions. This is a very delicate issue, with the overriding goal being to minimize the amount of time the designer must spend drawing a particle. If explicitly telling the system what to do takes less time than doing it in a way that the system understands, then explicitness is the better solution.

5 Conclusion

Designing visualizations for multi-valued, time-varying data is a very hard problem requiring many iterations of the design, implementation, and critique cycle. Furthermore, designs are traditionally done on paper, and not in the target medium. This works for some types of visualizations, but is much less effective when the final medium is an immersive display. Paper simply cannot capture the nuances of an immersive display as well as a design done in the target medium.

Our system provides the designer with a tool to quickly judge how well a particular design will work in the target environment. It is not designed to replace paper designs as it still takes longer to draw a design in our system than on paper, but it does allow the designer to preview a design before the costly step of coding it. As long as implementing a design is the costly step in completing a visualization, every effort should be made to reduce the number of times a design is implemented; our system is one step towards the goal of reducing the number of implementations to just one.

References

KEEFE, D., ACEVEDO, D., MOSCOVICH, T., LAIDLAW, D., AND LAVIOLA, J. 2001. Cavepainting: A fully immersive 3d artistic medium and interactive experience. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, 85–93.

SENAY, H., AND IGNATIUS, E. 1994. A knowledge-based system for visualization design. In *IEEE Computer Graphics and Applications*, 36–47.

SOBEL, J., FORSBERG, A., ZELEZNIK, R., LAIDLAW, D. H., PIVKIN, I., KARNIADAKIS, G., AND RICHARDSON, P. 2002. Particle flurries for 3d pulsatile flow visualization. In *IEEE Visualization Conference Poster Session*. in review.