Design Issues of Spray Rendering

Alex Pang, Naim Alper, Jeff Furman and Jiahua Wang
Computer and Information Sciences Board
University of California, Santa Cruz
Santa Cruz, CA 95064

ABSTRACT

We present the conceptual design and discuss the major issues of a very flexible and extensible visualization framework. This framework uses the metaphorical abstraction of a virtual can of spray paint that can be used to render data sets and make them visible. The idea behind this framework is based on behavioral animation and particle systems from computer graphics. Combining these two ideas, paint particles are endowed with intelligence so that they may seek out different features in the data set and manifest themselves visually. Different types of paint particles will generally result in different visualization techniques. We have used this framework to imitate existing techniques used in surface and flow visualization as well as in direct volume rendering. Aside from generalizing existing techniques, the framework also provides other advantages such as: (1) works with regularly gridded to scattered data sets; (2) handles both dense and sparse data sets; (3) allows selective progressive refinement; and (4) is modular, extensible and provides scientists with the flexibility of exploring relationships in their data sets in natural and artistic ways.

Key Words: Interactive Visualization, Particle Systems, Behavioral Animation.

INTRODUCTION

Today, there are many visualization techniques which provide users with different ways of looking at their data sets. Variations in these techniques may also arise depending on application specific needs. Different methods also exist for handling different data types (scalar, vector, tensor, higher dimensional) and structures (regular, rectilinear, curvilinear, scattered). For example, iso-surfaces from the marching cube algorithm [Lorensen-Cline87], surface shading from cuberille data [Chen-*et al.*85] and direct volume rendering [Drebin *et al.*88, Upson-Keeler88, Sabella88, Levoy90] are quite popular in molecular biology and medical imaging. On the other hand, particle tracing [Hin-Post93], stream lines and surfaces [Hultquist90, Helman-Hesselink91, van Wijk93] and streak lines [Lane93] are more popular in visualizing flow fields from computational fluid dynamics experiments. Users are also often provided mechanisms for animating and interacting with the data sets.

Another significant development in visualization is the availability of several commercial products such as AVS [Upson89], Iris Explorer [Sloane92] and Data-Explorer that provide users with an easy-to-use and extensible visualization environment. One of the major appeals of these products is their simple box wiring diagrams where users can graphi-

cally create a network of modules that transform data accordingly as they flow through these networks. As such, these environments all use the data flow approach. When a required module is not available, one can be built in a modular fashion and hence these systems are easily extensible. However, despite their popularity, these environments do have some drawbacks. Among the drawbacks of these systems include limited data types and the file oriented nature of data sets. None of the systems take advantage of powerful query tools available in current database systems [Stonebraker-*et al.*93]. In addition, users also encounter system limitations when dealing with very large data sets or dynamically changing data sets.

This paper presents an alternative visualization framework which preserves the ease-of-use and extensible nature of existing visualization environments and yet also addresses the limitations mentioned above.

In the next section, we present the conceptual design of spray rendering. We then discuss some of the design issues as well as how spray rendering can address the limitations brought up above. Finally, we present some examples and the directions of our continuing effort.

SPRAY RENDERING

What is it?

Spray rendering is a combination of two computer graphics techniques for modeling and animation [Pang-Smith93]. The first ingredient of spray rendering is particle systems which [Reeves83] originally developed as a technique for modeling natural phenomena. Particles are defined to have initial attributes such as color, trajectory and life span. These are then fired from some defined region and may spread and produce new particles of their own. Flames and fireworks are readily modeled with particle systems. In addition, if the path of these particles are integrated over time, then fields of grass can also be modeled. As mentioned earlier, particle systems have been used in the context of visualization where the particles are forced to interact with the data set as their environment. By adding another ingredient, one can obtain a much richer set of visual effects. The second ingredient of spray rendering is behavioral animation which was introduced by [Reynolds87] to manage the animation of a large number of actors. Examples that Reynolds presented included flocks of birds and schools of fish. Each member's behavior was dictated by its relative position in the environment and included effects such object avoidance. It was also not necessary for each member to know the whereabouts of everybody in its group. Thus, armed with the knowledge of the positions of a limited number of nearby friends, the species can exhibit behaviors such as group centering, and maintaining average speed and direction.

How does it work?

By combining particle systems and behavioral animation, particles are endowed with instructions to seek out specific target features in the data set and to react appropriately in a changing environment. Thus, the underlying mechanism behind spray rendering is the specification of different targets and behaviors for the particles.

To see how spray rendering can be a visualization tool consider the following. Rendering a data set is like painting. Given a data set, the rendering algorithm makes the set of numbers visible by assigning appropriate colors to the display that will faithfully mimic what the numbers are trying to represent. A crude equivalent to this process is pouring a bucket of paint over an invisible object in order to make it visible. The invisible object corresponds to the set of numbers that one is trying to visualize, while the rendering algorithm or the paint is the mechanism for making the data visible. One can also imagine using a paint brush or a can of spray paint instead. With the spray can, the user can aim the nozzle at the invisible object and selectively paint areas of interest by moving the can around. Thus, the name "spray rendering" is a result of our metaphor of providing users with virtual cans of spray paint for ren-

dering their data sets.

The power of this abstraction can be realized when one considers additional functions that these paint particles can do aside from sticking to invisible surfaces and highlighting those surfaces with the color of the paint. Since these particles are intelligent, we refer to them as smart particles or *sparts* for short. Visualization users who use spray rendering can picture themselves with an entire shelf of virtual spray paint cans loaded with these smart particles that can be applied to their data sets.

Components of spray rendering

There are two components of spray rendering: the spray can and the sparts.

The spray can is the delivery mechanism for getting the sparts into the data set. It is also a very intuitive metaphor so that most people can learn how to use it very quickly. The users can control several parameters of the can including the can position, orientation and contents. In addition, the user can also change the spart density (number of sparts released per dose), the distribution pattern of sparts and the shape of the can nozzle. Aside from the typical conical nozzle shape, the user may also specify a square nozzle, a line (rake) nozzle or a simple needle point nozzle.

The second component of spray rendering is the sparts. Just like ordinary particle systems, sparts keep track of their current state information consisting of position, age and trajectory. In addition, sparts also update their local set of data points as they move around the data set. More importantly, sparts may have an optional set of targets and behaviors. A wide variety of targets and behaviors may be customized as alluded to in the previous section. Aside from the visual behaviors of leaving a graphical primitive or a glyph [Ellson-Cox88] indicating that the target was found, a spart may also leave behind non-visible markers. This type of behavior may be used for different sparts to communicate with each other. Yet another property of sparts is a position update function which tells them where to move to in the next time frame.

Graphical user interface

Fig. 1 shows the main graphical user interface for spray rendering. The user can create multiple spray cans using the upper left panel buttons. Can parameters are adjusted by the middle left panel widgets. The main graphics window shows the user's current point of view which can be controlled by first selecting the camera icon on the bottom and then using the mouse to modify the view. The active spray can is similarly manipulated by first selecting the grab can icon on the bottom. To spray, the user simply selects the spray icon on the bottom and uses the mouse to pick and drag the 3D spray can. The lower left panel contains another graphics window. It shows what the active can sees and helps the users focus their spray better. Users may also control the can directly in this smaller window.

DESIGN ISSUES

Execution model

In coarse grained data flow environments such as AVS and Explorer, the modules making up the network of modules or the visual program consume and produce blocks of data. These environments are coarse grained because the block size is the same as an instance of the data model [Williams *et al.*92]. A problem with this approach is that when the data sets are large and the networks are complicated there is a serious growth in memory requirements because of data buffering and the performance suffers. A fine grained data flow environment has been proposed to alleviate this problem [Song-Golin93].

Our approach is not data flow oriented. Whereas in the data flow paradigm, data passes through in a stream-oriented fashion and are operated on by the modules, our functional building blocks operate on a limited region of the data set. We make use of the concept of sending multiple intelligent agents into the data set to look for features and display them.

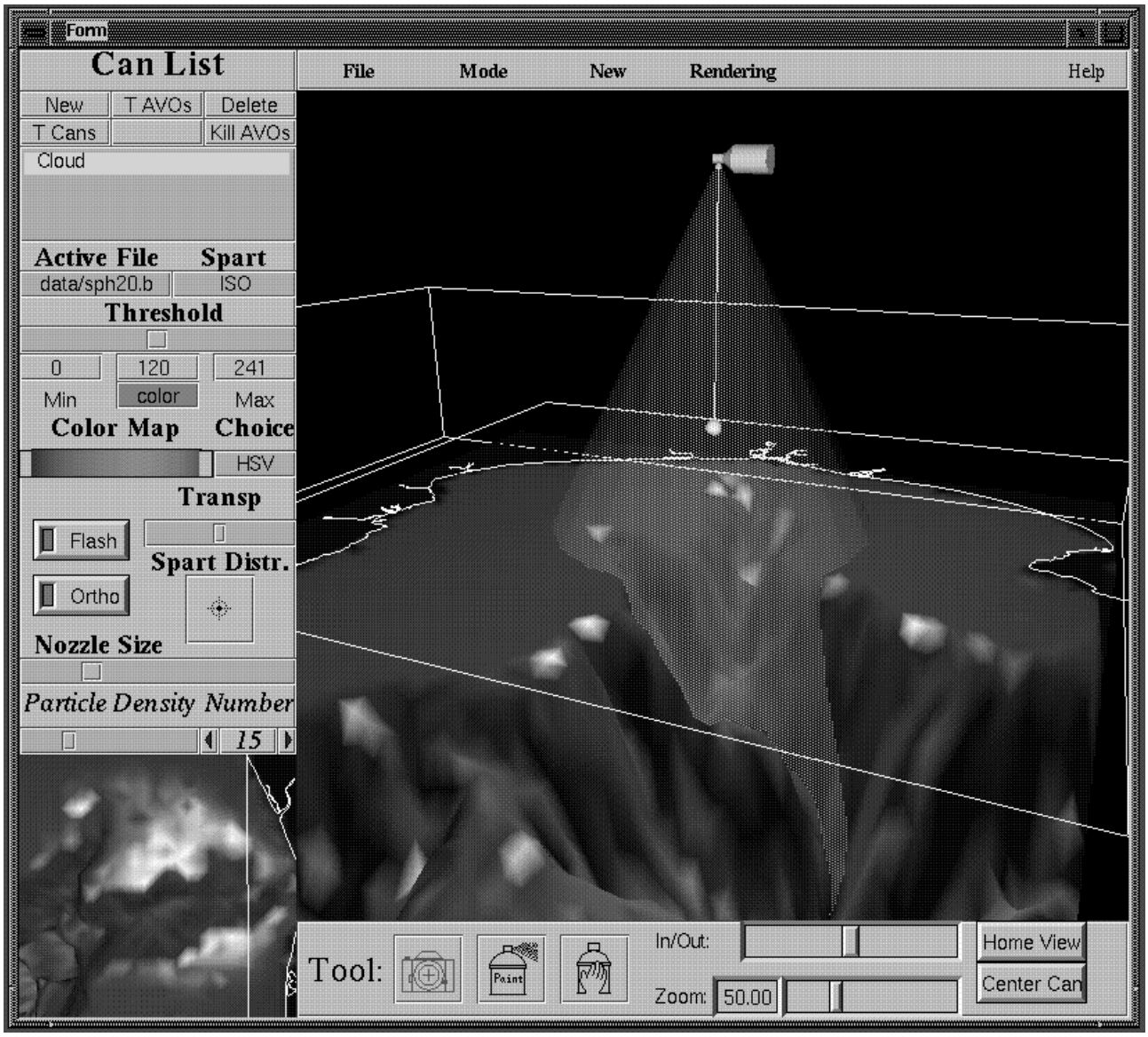

Figure 1: Graphical user interface for spray rendering.

Dealing with grids

There is no inherent requirement that the data to be visualized must lie within some grid system. Sparts have the notion of a local neighborhood. The range of this neighborhood is user definable and the data contained in it will change as the spart's position changes. The nature of the neighborhood is dependent on the nature of the data. If the data comes with an explicit or implicit grid structure, the neighborhood can be defined in terms of the computational space. If, on the other hand, the data is unstructured, the neighborhood can be defined in terms of Euclidean distances. Furthermore, for sparse data sets, a spart may extend its local domain to a larger region, or it may simply not manifest itself if there is insufficient local data.

Dealing with data types and formats

The sparts basically look for targets and exhibit a certain behavior depending on whether that target is found or not. This implies that the sparts need to know the data type that they are operating on. They handle different types according to the principle of polymorphism. For instance, if a spart is to calculate the maximum of a field in the process of a target search, it will intelligently call the appropriate routine depending on whether the field is a scalar or a vector field. If on the other hand the spart expects

a scalar field to do an iso-surface extraction and finds itself in a vector field, it will warn the user accordingly. If new data types need to be handled, the sparts can be modified or new sparts can be created to handle them.

Internally, sparts have their own data structures and data formats. New data formats are handled in one of two ways. One way, is a separate external program which converts the data format into something that spray rendering can handle. This would be suitable for handling common data formats such as NetCDF [Rew-Davis90]. Alternatively, one can extend spray rendering to handle the new data format by adding a new input routine. Eventually, we plan to integrate spray rendering with a database management system to take advantage of the more advanced data handling capabilities.

Dealing with large and dynamic data

When called to visualize large data sets, it is often not necessary to view the data set in its entirety. Unless direct volume rendering is called for, most parts of a large data set are occluded and are not visible. Even when the entire data set needs to be viewed, the limiting factor is often the screen resolution and our limited ability to digest all the details at once. Thus, one can often get away with a lower resolution rendering specially when this is coupled with animation.

Sparts can address the issue of interactively handling large data sets in two ways. But first observe that the complexity of spray rendering is dependent on the number of active sparts and the size of a spart's local neighborhood. If this neighborhood includes a substantial amount of data, then the performance of spray rendering will be slower. However, because sparts are not directly dependent on the size of the entire data set, spray rendering can allow the user to investigate very large data sets interactively by the following methods: (a) *Selective Focus.* Adjust the nozzle of the spray can so that it has a wide area of coverage. After the initial spray, selectively highlight regions of interest with a narrower beam of sparts. Note that because sparts look at their local neighborhood data points, they do not need to process the entire data set. (b) *Progressive Refinement.* Adjust the size of a spart's neighborhood or the size of the abstract visualization objects (AVO) [Haber-McNabb90]. Having a larger neighborhood to work with the spart may have a built-in smoothing operator. Alternatively, it may work on a small local neighborhood but produce a large AVO (e.g. sphere instead of a point) to represent the target that was found within that neighborhood. By adjusting these parameters, image quality is traded off with interactivity.

Dynamically changing data sets such as those found in turbulent fluid flow experiments can also be handled within the domain of spray rendering. Time is simply another parameter that the sparts use when seeking target features or deciding their next course of action. For example, flow paths such as those from streak lines are determined by the particle advection at the current time. Its next step is determined by the advection field in the next time frame, and so on.

Ease of use and extensibility

In order to make the system easy to use, we provide both a graphical user interface and a very intuitive metaphor of spray painting the data set. The operations of a spray can be learned very quickly and the only limitation is the users' imagination on what a spart can be designed to do.

There are two options to make spray rendering extensible. Either allow the users to build entire sparts (just as users can build entire modules in AVS and Explorer) or allow users to build sparts from components. In the latter case, we take advantage of more graphical user interfaces and extend our spray painting metaphor to include mixing pigments on a palette to get the desired color. There are two classes of users: those who know exactly what they are looking for in the data set and those who want to explore their data sets. For the former, we plan to provide an equation/relationship parser and for the latter, we provide a mix-and-match capability.

The ability to graphically create new sparts using mix-and-match is easily achievable once it is recognized that sparts can be broken down into more elementary components. A spart is made up of four basic components. *Targets* are what the spart is looking for in the data set and are functions that try to satisfy a boolean condition. *Behaviors* are usually made dependent on the targets and usually produce AVOs that are passed to the renderer. The other two components are functions that determine the new position of the spart and functions that determine whether a spart is to die or be spawned. Spart designers can compose new sparts from such functions to create different visualization techniques.

Intelligent queries/targets

Targets can be thought of as local feature extraction operators. They can be as simple as determining whether a data value at a point is within a certain range. Because they are boolean returning functions, complex features can be searched for by constructing complex boolean relationships. Also, a particular target function can be designed to parse complex mathematical expressions that the user enters interactively. For instance, this capability can be used to incrementally derive data from raw data. A rich vocabulary of primitive targets that are interoperable can lead to a very expressive facility to define what a spart is to look for. A particular combination can be saved as a macro so that it can be used as a primitive to facilitate ease of use. Thus, simple targets can be combined together to form more complex targets.

Arbitrary behaviors

Behaviors can be made dependent on the satisfaction of the target condition so that visual objects are produced where the conditions are satisfied. For instance, if one is looking for an iso-surface, polygons making up part of the surface will be generated only if the condition defining the surface is satisfied. However, behaviors need not always be visual. Markers can be deposited at those locations as well. These are non-visual behaviors that can facilitate complex communication between sparts over time. Such complex behavior constitutes the behavioral animation aspects of spray rendering and could be useful for achieving interesting visualization techniques.

The position update component of a spart can be deterministic or random. It can also be dependent on the data field such as particle advection in flow fields or gradient descent in scalar fields. The users may choose to define some other position update function as well. The same thing is true of death functions which determine when the spart is to die. Examples include a fixed number of time steps, going out of the data boundary, finding the target, etc. Even a small collection of these functions enriches the expressiveness of the system in achieving novel visualization techniques.

Encapsulating other algorithms

Spray rendering can take advantage of efforts elsewhere by encapsulating other algorithms within its framework. This is achieved by "localizing" the algorithm so that it is now viewed from the perspective of a spart. This process is not unlike the exercise taken in converting sequential programs to data parallel programs. For instance, in the marching cubes algorithm for iso-surfaces, all the cells in the volume are visited to determine the presence and orientation of a surface. To localize this algorithm, we pretend that the spart is traveling through the volume and therefore must do the same surface test for each cell in its path. Note that the localized version does not require all the cells of the volumes to be visited. Therefore there is a distinct possibility of holes in the resulting surface. This is examined more closely in the next section.

As described above, one can see that spray rendering can be easily extended. As new data types need to be incorporated or new targets or behaviors need to be implemented, new sparts can be programmed to handle them. Sparts offer a modular and extensible mechanism for adapting to the changing needs of the user.

ANATOMY OF A SPART

This section will analyze the different components of a simple spart. We also show how small variations in the spart's definition can lead to quite different visualization effects.

As an illustration, we examine the iso-surface sparts. These sparts have the same objectives as the marching cubes algorithm but have a different search path. Here, each spart is sampling the data volume defined by its path from the can to the first target surface it finds. The iso-surface spart consists of the following four components:

Target:
`IsoThresh[Data] (SurfFound) (Tag)`
Behavior:
`IsoSurf [Data] [SurfFound] [Tag]`
Position update:
`RegStep [Data]`
Death:
`OutOfVol [Data]`

In the composition above, inputs are indicated by square brackets and outputs by parentheses. All the components take the data set named `Data` as input. The target function `IsoThresh` determines whether the current cell contains a surface, specified as a threshold level by the user, and outputs the boolean `SurfFound`. The encoded tag for the lookup table is also output. These outputs are received as input by the `IsoSurf` behavior component. Whenever the `SurfFound` flag returns true, this function outputs the surface polygons using the tag and lookup table. The `RegStep` function takes regular steps along the initial direction of the spart. The step size is governed by a parameter. The `OutOfVol` death function kills the spart if its position falls outside the bounding box of the data set.

This particular composition will allow the sparts to find all the relevant iso-surfaces along its path within the data volume. A small variation in the death function will allow the spart to find only the front facing surfaces with respect to the sparts. This can be achieved by introducing a compound death function described by `OutOfVol[Data] or [SurfFound]`. Also notice that a similar small change to the position update function will allow traversal of all adjacent cells along the path as opposed to fixed step sizes along the path. This can be achieved by replacing the `RegStep` function with `NextCell` function.

It is quite easy to obtain a different visual effect from the same data set. Suppose the user wishes to paint the surface colors according to the steepness of the neighboring region. The `IsoSurf` function may simply be replaced with another function, say `GradientSurf[Data][SurfFound]`, which calculates the steepness of the surface around the cell and paints the surface as a function of steepness. The rest of the spart composition would still be the same. Yet, the visual effects can be quite different.

Because of the discrete sampling nature of these particles, surfaces will be generated only on the cells that have been hit. In this case, the user cannot be sure whether the absence of the surface in an area is due to undersampling or there is in fact no surface present. This is not particularly a problem since the users can choose to flood the data with sparts to ensure that all the cells are traversed. Alternatively, users can also modify the behavioral component such that aside from `IsoSurf`, it will also spawn off a new iso-spart in the vicinity of a previously discovered surface.

SAMPLE SPARTS

Here are some common and not so common sparts that one can easily identify with existing visualization techniques.

Surface seeking sparts

These sparts look for surfaces in the data set. The determination of what constitutes a surface is made locally by the spart. So, it can find more general surfaces than those present in the standard polygonal world. For example, a surface may be represented as a bilinear patch by a spart's four nearest points. Or a surface may be deemed to exist based on

the density of the data points in the spart's neighborhood. The behavior of the spart is not limited to highlighting the entire surface that it hits. It may simply display the intersection point. Alternatively, the spart can blot part of the surface with a paint spot, or it may just bounce off the surface in search of other surfaces.

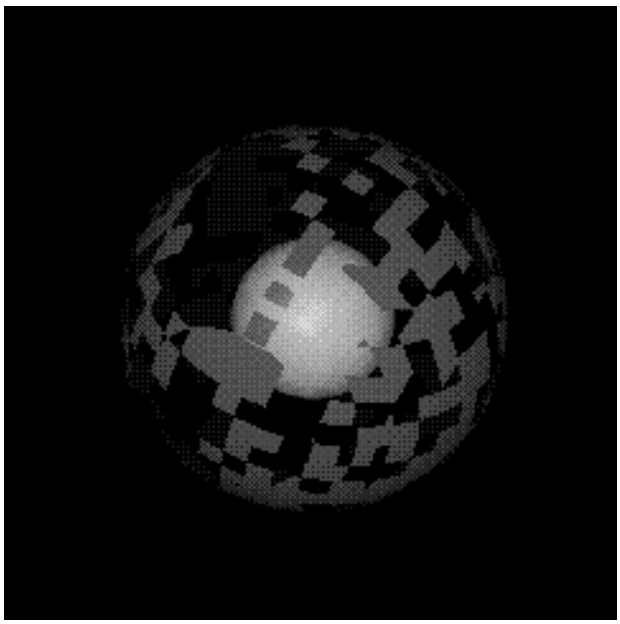

Figure 2: Two iso-surfaces of a synthetic volume where densities falls off uniformly away from the center. Outer iso-surface shows partially filled in effect.

Volume penetrating sparts

These sparts may not have specific targets. They may act like high energy particles bombarding the data sets. The visual effects of passing these sparts through the data set depend on their behavioral description. Since the direction of the spart's path does not have to coincide with the viewer's gaze, one can generate view dependent effects which highlights the internal structure of the data. The dust sparts in Fig. 3 simply color their positions according to the data values that they encounter.

Flow tracking sparts

Flow tracking sparts are ideal for visualizing vector fields. These sparts typically do not have specific targets and usually do not have

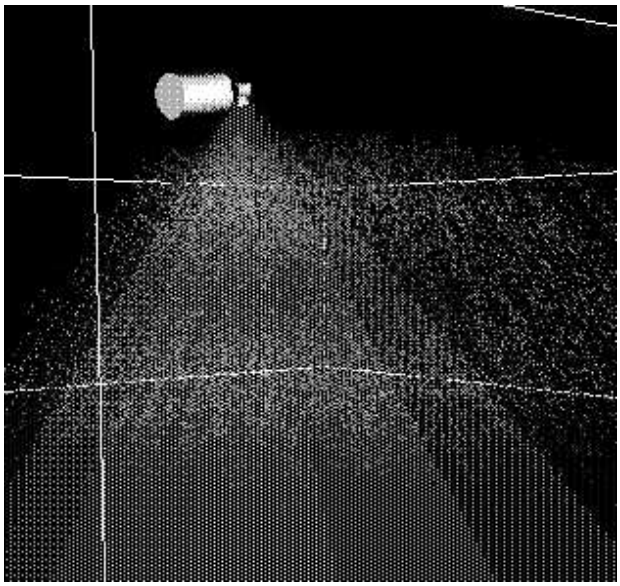

Figure 3: Dust particles in synthetic volume. Bounding box and spray can are visible.

an initial velocity or trajectory. Instead, they are introduced into vector fields where they are influenced and carried around by the surrounding neighboring forces. The phenomena of interest are usually the flow patterns rather than surfaces. Therefore, these sparts manifest themselves by leaving a trace of their path as they advance from one state to another. Flow tracking sparts may work in pairs or groups so as to form flow ribbons and rakes respectively. Different integration routines may be plugged in for calculating particle advection. Finally, these sparts may also be modified to do streak lines in time dependent flows.

Meta-sparts

Meta-sparts are slightly different when compared to the previous sparts because their targets are not based on the original data set. Instead, meta-sparts seek out markers left behind by other sparts. An example of a meta-spart is a garbage collecting spart that simply removes the visual cues from the rendered image. Typical uses for such a spart include editing, cleaning the rendered scene and reducing the overall scene complexity. With meta-sparts, one can also create sparts that produce secondary effects by combining results of previous sparts. It is important to

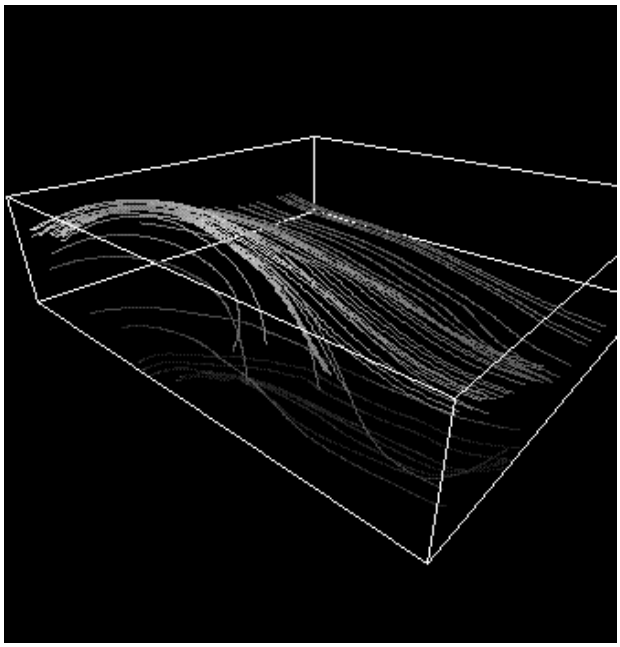

Figure 4: Stream lines in a simulated wind vector field.

note that the original data set is left untouched by this and all other types of sparts.

SUMMARY

Spray rendering provides a framework for applying diverse visualization techniques in a unified manner. It is born out of a need for scientists to be able to visualize and explore large data sets with widely varying data structures without learning several different packages. Because of the properties of sparts, novel visualization effects can be easily designed and tested. The interface for launching sparts appears intuitive and encourages users to interactively explore their data sets.

We have shown that spray rendering has great potentials and is also practical. Practicality comes about by providing user adjustable parameters to trade-off interactivity against image quality. We are currently expanding spray rendering to include the mix-and-match capability, 3D VR interface and also to a multi-media collaborative visualization system.

REFERENCES

Chen, L.S., *et al.*, 1985. Surface shading in the cuberille environment. *IEEE Computer Graphics and Applications*, 5(12), pp. 33–43.

Drebin, R., Carpenter, L., Hanrahan, P. 1988. Volume rendering. *Computer Graphics*, 22(4), pp. 65–74.

Ellson, R., Cox, D. 1988. Visualization of injection molding. *Simulation*, 51(5), pp. 184–188.

Haber, R.B., McNabb, D.A. 1990. Visualization idioms: A conceptual model for scientific visualization systems. In G. M. Nielson, B. Shriver, Rosenblum, L. J. 1990, editors, *Visualization in Scientific Computing*, pages 74–93. IEEE Computer Society Press.

Helman, J.L., Hesselink, L. 1991. Visualizing vector field topology in fluid flows. *IEEE Computer Graphics and Applications*, 11(3), pp. 36–46.

Hin, A.J.S., Post, F.H. 1993. Visualization of turbulent flow with particles. In *Visualization'93 Proceedings*, pages 46–52.

Hultquist, J. 1990. Interactive numerical flow visualization using stream surfaces. Technical Report RNR-90-009, NASA Ames Research Center.

Lane, D.A. 1993. Visualization of time-dependent flow fields. In *Visualization'93 Proceedings*, pages 32–38.

Levoy, M. 1990. A hybrid ray tracer for rendering polygon and volume data. *IEEE Computer Graphics and Applications*, 10(2), pp. 33–40.

Lorensen, W.E., Cline, H.E. 1987. Marching cubes: A high-resolution 3d surface construction algorithm. *Computer Graphics*, 21(4), pp. 163–169.

Pang, A., Smith, K. 1993. Spray rendering: Visualization with smart particles. In *Visualization'93 Proceedings*, pages 283–290.

Reeves, W.T. 1983. Particle systems - a technique for modelling a class of fuzzy objects. *Computer Graphics*, 17(3), pp. 359–376.

Rew, R., Davis, G. 1990. Netcdf: an interface for scientific data access. *IEEE Computer*

Graphics and Applications, 10(4), pp. 76–82.

Reynolds, C.W. 1987. Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics*, 21(4), pp. 25–34.

Sabella, P. 1988. A rendering algorithm for visualizing 3d scalar fields. *Computer Graphics*, 22(4), pp. 51–55.

Sloane, G. 1992. *IRIS Explorer Module Writer's Guide*. Silicon Graphics, Inc. Document Number 007-1369-010.

Song, D., Golin, E. 1993. Fine-grain visualization algorithms in dataflow environments. In *Visualization'93 Proceedings*, pages 126–133.

Stonebraker, M., *et al.*, 1993. Tioga: A database-oriented visualization tool. In *Visualization'93 Proceedings*, pages 86–93.

Upson, C., Keeler, M. 1988. V-buffer: Visible volume rendering. *Computer Graphics*, 22(4), pp. 59–64.

Upson, C. 1989. The application visualization system: A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, 9(4), pp. 30–42.

Wijk, J.J.van 1993. Flow visualization with surface particles. *IEEE Computer Graphics and Applications*, 13(4), pp. 18–24.

Williams, C., Rasure, J., Hansen, C. 1992. The state of the art of visual languages for visualization. In *Visualization'92 Proceedings*, pages 202–209.