Spray Rendering

Alex Pang
Computer and Information Sciences Board
University of California, Santa Cruz

Spray rendering is a framework for creating and experimenting with different visualization techniques. The name spray rendering is derived from the metaphor of using a virtual spray can to paint data sets. Depending on the type of 'paint' in the can, data can be highlighted in different ways. Spray rendering is not limited to this particular metaphor. In fact, other metaphors that have also been used include a flashlight and a probe. Thus, when we speak of spray rendering, we are referring to the localized nature of the visualization algorithms and the manner in which the algorithms are 'sent' to the data sets. Several advantages arise from looking at visualization algorithms this way. Among these are extensibility, grid independence and ability to handle large data sets. This paper presents the benefits, conceptual design, issues and the directions of spray rendering.

Background

The goal of spray rendering is to provide a framework for exploring different visualization techniques. To achieve this goal, spray rendering must be able to encapsulate existing visualization methods (e.g. marching cubes [1], particle tracing [2], stream lines and surfaces [3], streak lines [4], etc.) and also be able to grow and accommodate new and yet to be developed methods. The plan of attack is simple: find out the strategy behind most visualization algorithms and make sure the system is extensible.

Visualization algorithms are designed to highlight features found in data sets in different ways. They can be viewed as having a two-step process: feature extraction followed by an appropriate display of those features. Features may include the presence and location of iso-valued areas, the direction and magnitude of flow fields, and the distribution of densities, to name a few. Once these features are extracted, they may be visually represented using contour lines, isosurfaces, vector glyphs, animated particle traces and ribbons or as accumulated density fields as viewed by the user. Using this simple generalization, existing visualization algorithms can be analyzed into different components of feature extraction and visual display processes. Spray rendering builds upon this simplistic view of visualization algorithms by allowing users to encapsulate existing algorithms and to create new visualization algorithms by combining different components together.

In recent years, one of the significant development in visualization is the availability of several commercial products such as AVS, Iris Explorer, Khoros, 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 graphically 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. 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 benefits and 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 does it offer?

Spray rendering offers several advantages. The list includes: (1) *grid independence.* It works with regularly gridded to sparse scattered data sets. (2) *large*

data sets. It does not require the entire data set to be in memory since it works on a local subset at a time. Spray rendering allows selective focus and progressive refinement and is ideal for exploratory visualization applications. (3) *extensibility.* It encompasses most visualization techniques and allows users to easily create and explore new ones. (4) *ease of use.* It provides users with the flexibility of exploring relationships in their data sets in natural and artistic ways.

What is it?

Spray rendering is a combination of two computer graphics techniques for modeling and animation [5, 6]. The first ingredient of spray rendering is particle systems [7] which was 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. While particles systems have been used to model fireworks and grass, they have also been used in the context of visualization where particles are forced to interact with the data set [8, 9]. 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 [10] 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 local 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. The latter allows the user easier control on which area of the data to visualize.

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" in the sense that they have specific targets and corresponding behaviors, we refer to them as smart particles or *sparts*. Visualization users who use spray rendering can picture themselves with an entire shelf of virtual spray paint cans loaded with different types of sparts 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 associated with the can such as the 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 spart. 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 space. More importantly, sparts may have an optional set of targets and behaviors. Aside from the visual behaviors of leaving a graphical primitive or a glyph indicating that the target was found, a spart may also leave behind non-visible markers. This type of behavior may be used by different sparts to communicate with each other. Yet another property of sparts is a position update function which tells them where to move in the next time frame, and a spawn/death function which tells them when to spawn new sparts or to terminate themselves.

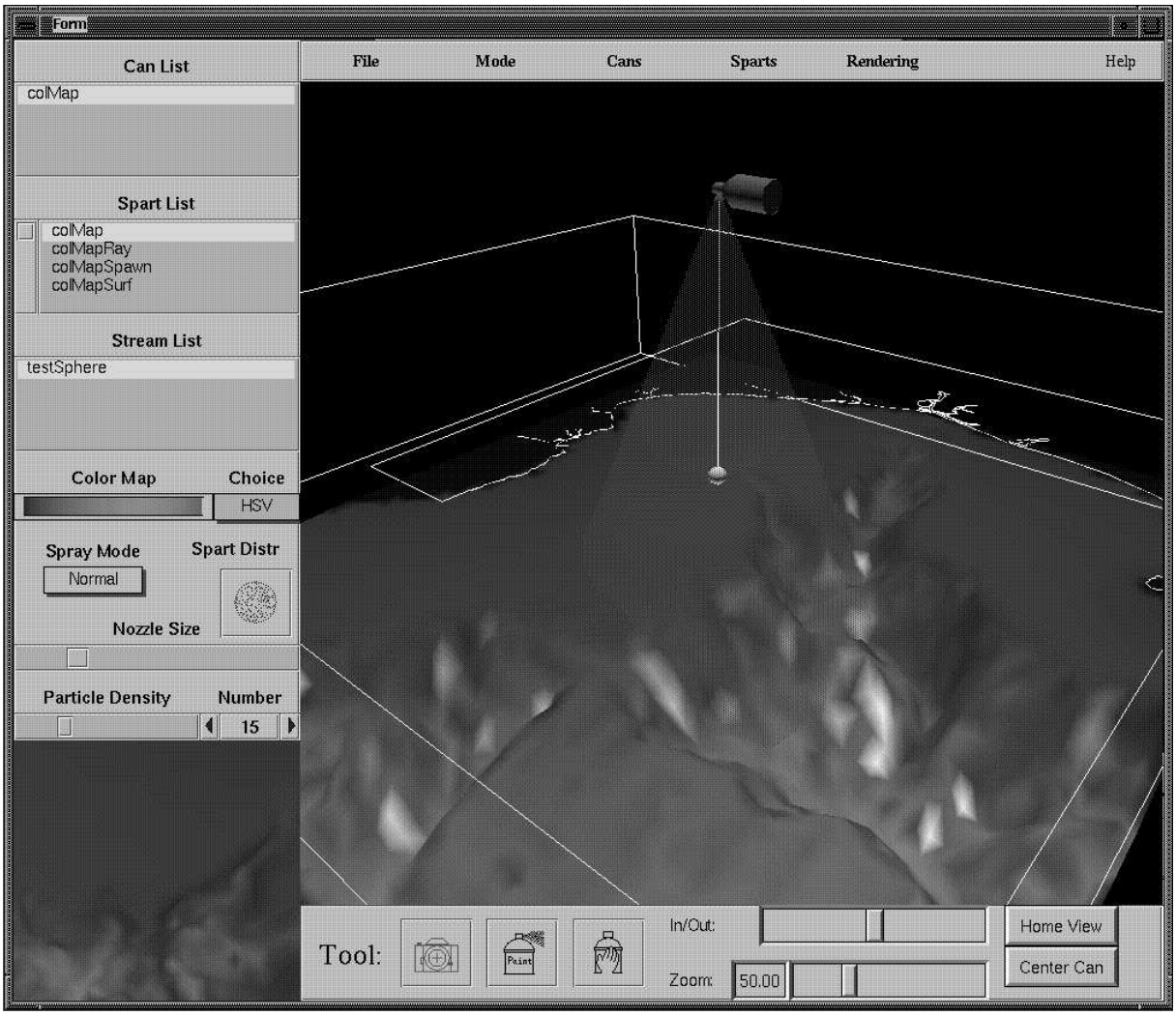

Figure 1: The main graphical user interface for spray rendering. Users select the type of sparts and create a new spray can. Multiple spray cans with different sparts may be created. The main graphics window shows the user's current point of view which can be controlled by first selecting the camera 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 which shows the view from the active can. 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 of the visual program network 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 [11]. 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 hence the performance suffers. A fine grained data flow environment has been proposed to alleviate this problem [12].

Instead of a data flow oriented execution model, where data flow through and are executed upon by modules, our functional building blocks operate on a limited region of the data set. Conceptually, we are sending multiple intelligent agents into the data set to look for features and display them.

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 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. The first 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. Alternatively, one can extend spray rendering to handle the new data format by adding a new input routine.

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 utilize a lower resolution rendering, especially when this is coupled with animation.

Sparts can address the issue of interactively handling large data sets in one of 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 rather than the size of the data set. If this neighborhood includes a substantial amount of data, then the performance of spray rendering will be slower. Thus, 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) [13]. 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 operation of a spray can is very easy to learn 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 extend our spray painting metaphor to include mixing pigments on a palette to get the desired color. New sparts can be created interactively by simply mixing and matching different spart components together. This 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. Typically, a target function returns a boolean value and the accompanying data representing the target features. Complex target features can be constructed using compound boolean relationships of simpler target functions. 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.

Arbitrary behaviors

Behaviors, such as visual objects, are produced when and where target 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. For example, markers can be used to combine information from multiple data sets.

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 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

Not all visualization algorithms can be encapsulated nicely into spray rendering. Particularly difficult ones are view-dependent algorithms, such as direct volume rendering, since the viewpoint of the user and the spray can do not have to coincide. One possible way of incorporating them is to make views from the spray can be part of the AVO list.

Fortunately, 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 similar to the exercise taken in converting sequential programs to data parallel programs. For instance, in the marching cubes algorithm, 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 (see Fig. 2). In addition, some of the cells may be visited more than once and thereby duplicating efforts of previous sparts. To guarantee complete coverage and produce the same results as its global counterparts, one of the spray can adjustments is to flood the data with sparts (again see Fig. 2).

As new data types need to be incorporated or new targets or behaviors need to be implemented, new components can be programmed to handle them. This is simpler than programming the larger modules found in data flow visualization environments. Sparts, therefore, offer a modular and extensible mechanism for adapting to the changing needs of the user.

ANATOMY OF A SPART

This section analyzes the different components of a simple spart and shows how small variations in the spart's definition can lead to different visualization effects.

As an illustration, we examine the iso-surface sparts. These sparts and the marching cubes algorithm have the same objectives and output the same polygons in cells that they visit. However, they differ in their search path. While marching cubes visit each cell in a systematic order, sparts visit only those cells that they encounter along their initial trajectory from the can. The iso-surface spart consists of the following four components:

```
IsoThresh[Data] (Found) (Tag) (IsoVal)
IsoSurf  [Data] [Found] [Tag] [IsoVal] (Obj)
RegStep  [Data]
OutOfVol [Data]
```

In the composition above, input fields are indicated by [ ] and output fields by ( ). Aside from input fields, functions may also have user adjustable parameters such as iso-values, step-sizes, etc. `IsoThresh` is a simple *target* function that examines its current location within the input stream `Data`, which is bound to a scalar data file, for the iso-value parameter. It outputs the boolean `Found`, an index to a lookup table `Tag` that indicates the configuration of the cell vertices, and the iso-value parameter `IsoVal`. If `Found` is true, the *behavior* function `IsoSurf` will generate one or more polygon visualization objects `Obj` in places specified by the `Tag` field. The spart then advances a fixed step size, according to the parameter set in the *position update* function `RegStep`, along its initial direction and checks to see if it has exited the data space using the *death* function `OutOfVol`. These functions are repeated until the spart is terminated after exiting the bounding box of the data volume.

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 spray can. This can be achieved by introducing a compound death function described by `OutOfVol [Data] or [Found]`. 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. We can simply substitute the behavior function with, say `GradientSurf [Data] [Found] [Tag] [IsoVal] (Obj)`, which calculates the steepness of the surface around the cell and colors the surfaces as a function of steepness. The rest of the spart composition would still be the same. Yet the visual effects can be quite different.

OTHER METAPHORS

The spray can metaphor is natural and easily learned by most people. We have also extended the spray can to two other metaphors which are also quite intuitive to use and require little deviation from the spray rendering design. These are briefly described below.

Flashlight. Instead of sparts "sticking" to data features, we allow them to "slide" off and disappear as the user moves the flashlight away from that direction and reappear when the flashlight is pointed there again. Alternatively, the user can hold the flashlight steady, but aim it at a dynamic or "moving" data and watch the changes go by. The same sparts used in spray cans can be used in flashlights. A relatively inexpensive way of keeping track of what is currently visible from the flashlight is with a first-in-first-out AVO queue.

Probe. Probes allow users to examine a specific region in the data space. As the user points the probe at a different location, features in that area are displayed. Thus, probes are simply spray cans operating in flashlight mode with the additional restriction that only one spart is being fired at a time (e.g. Fig. 3).

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 (e.g. Fig. 2). 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 trilinear patch by a spart's eight 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.

Flatland sparts

A common technique that scientists use in analyzing data is to display 2D profiles. This can be generalized to arbitrary planar cross-sections of the data. We have therefore designed some sparts that will travel on a plane that is normal to the spray can's line of sight and a fixed distance away. These sparts can resample values from neighboring points and use pseudo-coloring to represent the 2D field. Alternatively, they can also track contours on that plane. Fig. 3 show these two in action.

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.

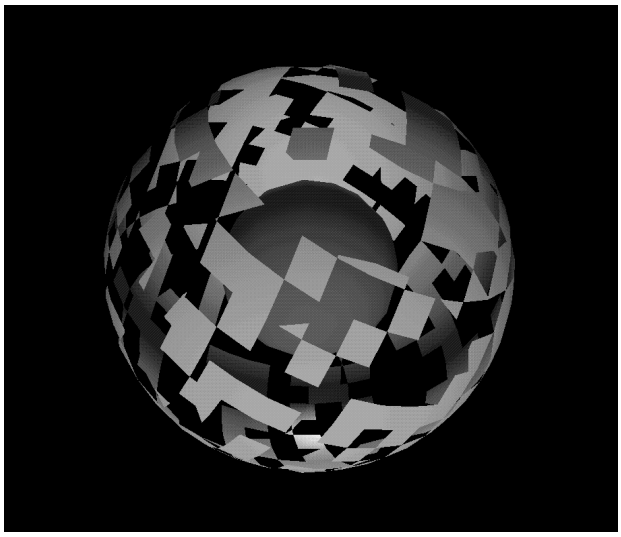

Figure 2: Two iso-surfaces of a synthetic volume where the density field falls off uniformly away from the center. The outer iso-surface shows the partially filled in effect of an exploratory spray. The complete inner iso-surface is obtained by flooding the volume with sparts.

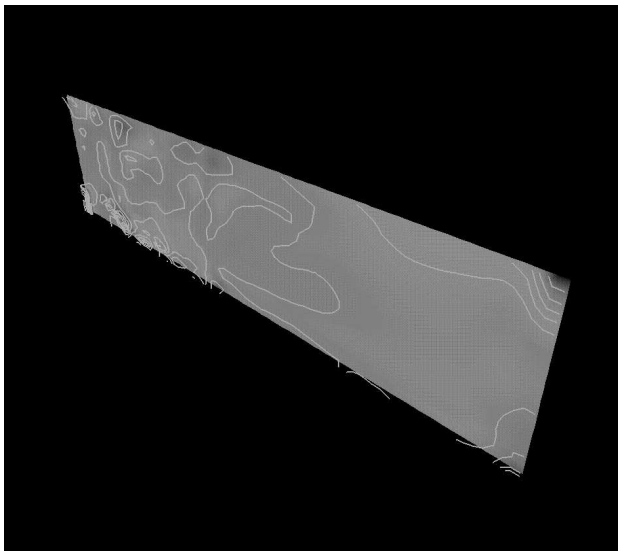

Figure 3: A pseudo-colored cross section and the contours on an adjacent cross section Each is obtained by a single flatlander spart in probe mode. A different cross section is obtained when the user moves the probe around. Cross sections can be constrained to primary planes or may be unconstrained.

Since the direction of the spart's path does not have to coincide with the viewer's gaze, one can generate view-independent effects which highlight the internal structure of the data. The dust sparts in Fig. 4 simply color their positions according to the data values that they encounter. If flashlight mode was used instead of spray mode, only those dust particles directly in the can's cone of fire would be visible.

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 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 (e.g. Fig. 5). Flow tracking sparts may work in pairs or groups so as to form flow ribbons and rakes respectively. These sparts may also be modified to do streak lines in time-dependent flows. Finally, instead of leaving leaving path traces, animation may also be used to obtain a better idea on relative speed magnitudes.

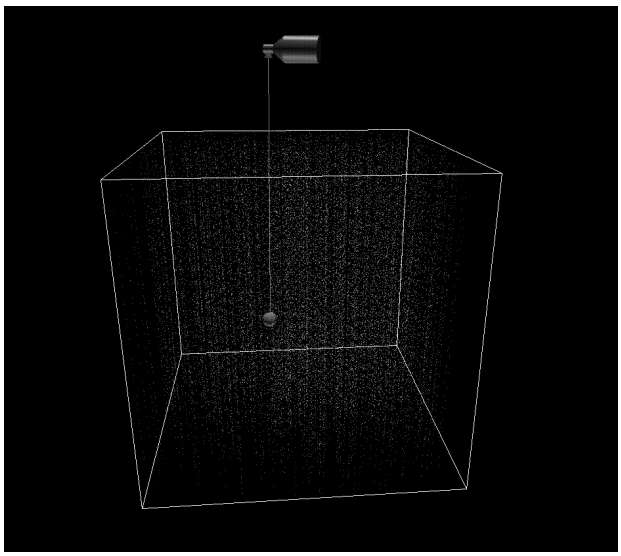

Figure 4: Dust particles in synthetic volume. Bounding box and spray can are visible.

PERFORMANCE

The performance of spray rendering depends more on the number and complexity of sparts more than the size of the data sets. We found that the limiting factor has been the display of the AVOs that are generated by

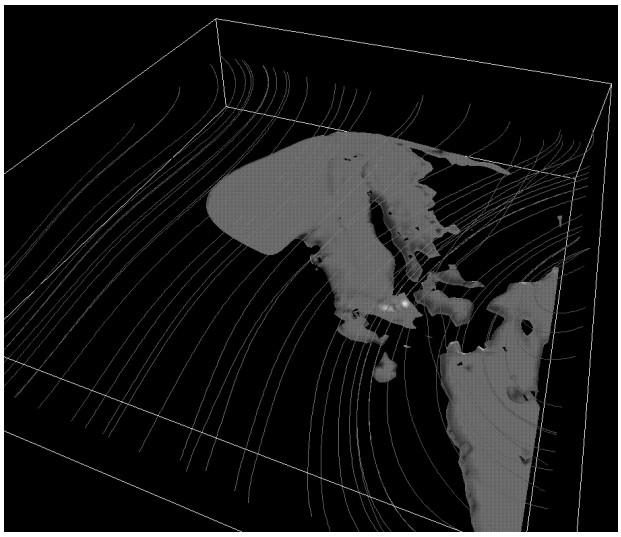

Figure 5: Stream lines in a simulated wind vector field.

the sparts. Thus, we also allow the users to toggle the display of AVOs from a spray can to reduce clutter and to improve interaction. Our development and testing of spray rendering is based on SGI platforms that range from Indigos to the Reality Engine 2. We find the display rate of about 300K shaded polygons per second to be adequate for handling the AVOs from generated from a couple of spray cans.

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. Users can employ quick exploratory sprays or complete data traversal with flood fills. 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. The extension to flashlight and probe modes also appear to be quite useful.

We see great potentials in spray rendering and are currently expanding it to handle irregular grids, to include 3D VR interface, and to provide multi-media collaborative visualization capabilities.

ACKNOWLEDGEMENTS

The contributions from the following are greatly appreciated: Naim Alper, Jeff Furman, Tom Goodman, Elijah Saxon, Craig Wittenbrink, Peter Dommel, Jiahua Wang and Kyle Smith. We would also like to thank Dr. Teddy Holt and Dr. Paul Hirschberg for kindly providing us with some of the data sets used in the figures. The comments by the reviewers are gratefully acknowledged and incorporated within the page limitations of this paper. This work is funded in part by ONR grant N00014-92-J-1807 and NSF grant CDA-9115268.

References

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

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

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

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

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

[6] A. Pang, Naim Alper, Jeff Furman, and Jiahua Wang. Design issues of spray rendering. In *Compugraphics'93*, pages 58–67, 1993.

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

[8] G. D. Kerlick. Moving iconic objects in scientific visualization. In *Proceedings: Visualization '90*, pages 124 – 129. IEEE Computer Society, 1990.

[9] K. Sims. Particle animation and rendering using data parallel computation. *Computer Graphics*, 24(4):405 – 413, 1990.

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

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

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

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

Biography

Alex Pang is an Assistant Professor in the Computer and Information Sciences Board at the University of California, Santa Cruz. He obtained his BS in Industrial Engineering from the University of the Philippines, and received his MS and PhD in Computer Science from UCLA in 1984 and 1990. His research interests are in computer graphics, scientific visualization, collaboration software, multimedia and virtual reality interfaces.