Spray Rendering:
A New Framework for Visualization

Alex Pang and Kyle Smith

UCSC-CRL-93-01
January 1993

Board of Studies in Computer and Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064

email addresses: pang@cse.ucsc.edu, kyle@cse.ucsc.edu
Supported in part by Office of Naval Research grant no. N-00014-92-J-1807

Abstract

We propose a new framework for doing scientific visualization that allows the users to freely explore their data set. This framework uses a metaphorical abstraction of a virtual can of spray paint that can be used to render data sets and make them visible. Different cans of spray paint may be used to color the data differently. Different types of spray paint may also be used to highlight different features in the data set. To achieve this, individual paint particles are endowed with intelligent behavior. This idea offers several advantages over existing methods: (1) it generalizes the current techniques of surface, volume and flow visualization under one coherent framework; (2) it works with regular and irregular grids as well as sparse and dense data sets; (3) it allows selective progressive refinement and can be implemented on parallel architectures in a straight forward manner; (4) it is modular, extensible and provides scientists with the flexibility for exploring relationships in their data sets in natural and artistic ways.

1. Overview

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 further imagine holding a can of spray paint, aiming the nozzle at the invisible object, and selectively painting areas of interest by moving the spray can around.

Spray rendering uses the metaphorical abstraction of providing the visualization user with virtual cans of spray paint for rendering their data. The power of this abstraction can be realized when we consider what additional functions these paint particles can do aside from sticking to invisible surfaces and highlighting those surfaces with the paint. Arbitrary characteristics can be assigned to the spray paint. Therefore, instead of painting the object using the color of the paint, one can imagine a formula X spray paint for highlighting specific surface temperatures. That is, these paint particles would behave differently depending on the data that they encounter. Furthermore, one can create a formula Y spray paint that will be activated only when it encounters a certain range of surface temperatures. Both of these particles are endowed with the ability to seek out certain target features in the data set. Once those target features are identified, the particles then behave or manifest themselves in various ways. Since these paint particles are smart, 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 that can be applied to their data sets (see Figure 1). Depending on the type of spart within the can, data will be rendered to highlight different features.

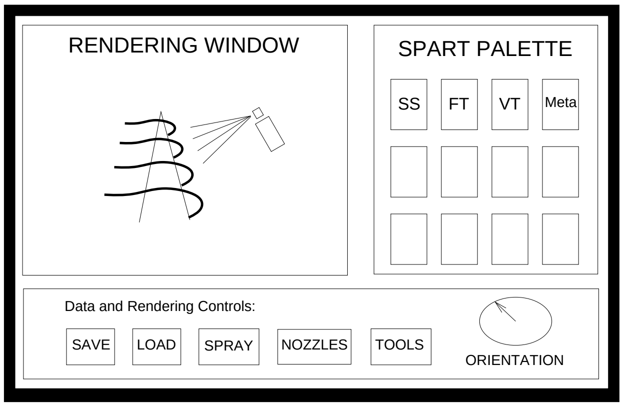

Figure 1. A mock panel showing a palette of sparts that the user selects from to fill the spray can. Data are then selectively visualized as the user moves the virtual spray can around the data set.

2. Challenges to current visualization techniques

In recent years, there has been significant advances in visualization techniques. A major driving force is fulfilling the flow visualization requirements usually found in Computational Fluid Dynamics (CFD) experiments. Examples of techniques that came out of this effort include: particle traces along trajectories, streams and ribbons, [Hult90, Helm91] as well as volume visualization of vector field variables. These are often incorporated in large visualization projects developed at the nation's leading research laboratories such as those reported in [Phil90]. Some of the more recent work in flow visualization include a virtual environment interface [Brys92], an interactive flow visualization using color lookup tables [VanG92], and extending direct volume visualization from scalar to vector fields [Craw92].

Another driving force in visualization is medical imaging. Non-invasive techniques for data gathering produce cross-sectional views that must be reconstructed (*e.g.* curved coronal views of the lumbar spine [Rhod85]) to facilitate understanding of the underlying three dimensional structure and aid in pre-surgical planning [Gold85]. Techniques for viewing three dimensional data volumes have evolved from image processing and data conversion. A few examples would be extracting contour edges and surface tiling [Fuch77], finding isosurfaces with the marching cubes algorithm [Lore87] and 3D surface shading in the cuberille and voxel environment [Chen85, Kauf88, Levo88], and finally to direct volume visualization that simulates the amount of material traversed by a light ray in an opacity parameter [Dreb88, Upso88, Sabe88]. More recent work in this area are geared towards extending volume rendering to work with non-rectangular grids such as curvilinear grids commonly found in CFD data [Wilh90]. Significant work is also aimed at speeding up volume rendering either through hierarchical organizations [Laur91] or parallel computation such as those reported in the 1992 workshop on volume visualization held in Boston. Because data come in different forms and from different disciplines, a convenient way of handling this apparent inhomogeneity is through a data flow model where data is transformed as it goes through different modules in the network. Once this abstraction is made, graphical user interfaces can be built to provide a relatively easy framework for interacting with the data and to offer a flexible, customized solution to a wide variety of applications. Recent advances in visualization platforms such as AVS, apE, Khoros and Explorer which incorporate very user friendly interfaces have all contributed to the wider acceptance of visualization tools by the scientific community.

Overall, there has been significant progress in different areas such as computer graphics, vision, image processing, human computer interfaces and computational sciences during the last few years. The NSF panel report on visualization in scientific computing [McCo87] served to bring these disciplines together in a focused and concerted effort at pushing the frontiers of visualization forward. Since then, the synergy between these different disciplines and application domains such as CFD, medicine, molecular modeling have produced a reciprocal relationship where visualization may help lead to new discoveries which in turn help drive the need to improve visualization technology. However, there are still some major obstacles to be addressed. For example, most of the visualization work to date has dealt with data rich environments where errors introduced by interpolation to fill in holes in the data set were at an acceptable level. In data poor environment, data are sparsely distributed and often in unstructured, irregular grids. An initial work along this direction that handles unorganized data points is already being undertaken [Hopp92]. Another consideration is that data may be noisy and users must be given a visual assessment of data quality at different spatial locations. Still another challenge is to provide interactive steering capabilities to computational problems. This requirement places strong demands on the computational power, communication bandwidth, intelligent hierarchical organizations and difficult tradeoff decisions.

3. Spray rendering

There are several concerns with current visualization techniques that spray rendering can address. First of all, most current rendering techniques are designed to display data of a particular type and generate a specific type of image (*e.g.* polygonal surfaces or volume rendering). Hence, users who need to visualize different data types or combine different visual effects are forced to learn different packages. Secondly, most visualization packages only deal with dense data on either regular or curvilinear grids. There is a need for a single platform that deals with dense data on regular and curvilinear grids, as well as sparse unstructured data. Thirdly, most visualization packages require a pre-processor to do feature extraction that seeks out relationships in the data set. Typically, the features from the entire data set are extracted and then passed on to the renderer. This approach does not provide an interactive interface where less promising searches can be quickly discarded and efforts redirected to the more interesting paths. Finally, the visualization package should grow gracefully with the visualization needs of the users, and be extensible to new data types.

3.1. The paradigm

The underlying mechanism behind spray rendering is the specification of sparts with different targets and behaviors. To a certain extent, this technique is a combination of behavioral animation [Reyn87] and particle systems [Reev83, Sims90, Szel92] in that each particle is endowed with instructions on what to seek out and how to react to its changing environment. Particle systems were originally developed as a means of modeling natural phenomena such as fire, forests and grass that would have been difficult or nearly impossible with classical polygonal representation. The novel feature of particle systems is the procedural method of specifying the models. For example, particles are defined to have initial attributes such as color, trajectory and life span. Particles are then fired from some defined region and may spread and/or produce new particles of their own. Already, visualization researchers are using similar ideas in fluid flow visualization where tracer particles are injected and their paths traced. Behavioral animation, on the other hand, found its initial utility in computer animation applications by specifying group behavior such as those found in schools of fish or flocks of birds. Typically, some heuristics are imposed on individual characters to determine how the group as a whole will behave. Examples usually include avoiding collision, maintaining average speed and direction, as well as not straying too far away from the group center. More recently, interaction of the characters with their environment has been incorporated. An example of this is the leaves in the wind simulation [Wejc91]. Although both particle systems and behavioral animation were originally designed for modeling and animation applications, the combination of these two ideas provide a synergy with very promising contributions in the context of a framework for visualization.

The power of this framework can be realized if one takes a closer look at the different possibilities for defining individual sparts. First of all, it is convenient to think of sparts as small spherical balls or light emitting points. However, the individual sparts may also take the form of glyphs [Ells88] or 3D icons. For example, one can create paint particles that are shaped like leaves. These particles can then be sprayed or thrown into the wind to highlight wind flow and the presence of vortices. Other attributes that can be attached to a spart are the target features it is programmed to seek out within its local data set. A simple example would be sparts that seek out surfaces. A more interesting target would be one that shows relationships among different variables. For instance, a spart may be defined to manifest itself only if the wind speed is between 15 and 30 knots, air temperature is above 70 degrees Fahrenheit and irradiance is at least 300 watts/m^2 (*i.e.* good sailing conditions). Yet another attribute of a spart is its behavior once a target is identified. This is undoubtedly the most flexible and powerful component of a spart. For example, surfaces can be highlighted using the spart color or the surface color. The behavior may also be modified so that instead of sticking to the first surface that a spart encounters, it bounces off thereby simulating highly reflective surfaces. Alternatively, x-ray like sparts can be created that penetrate through the data and accumulate density and color information resulting in fuzzy, transparent, volume rendered images. Sparts can also be defined to look for clean data or differences in data compared to historical or calibrated data sets and effectively highlight data quality. In addition, sparts can be instructed to alter their own appearance or color as they move around the data set and/or leave a trail of their path. These sparts can be used to show flow patterns in vector data where the paths of the sparts are influenced and advected by the local forces within the data set. Likewise, they can be used to search and highlight iso-potential fields.

Sparts are dynamic entities. Thus, another simple extension would be to change the color of the path that a spart traces out to indicate the age of the spart. New sparts may be born, and older ones may be extinguished. Sparts may either work individually, which facilitates asynchronous parallel implementation, or they may work cooperatively and share information about their local surroundings. Therefore, in addition to defining spart to data interactions, one can also specify spart to spart interactions. Looking at the leaves example again, spart to spart interactions correspond to the collision of leaves. The amount of collision may then be mapped to rustling sounds or visual cues signifying the intensity of the wind. Spart designers also have the flexibility of organizing simple sparts into groups and hierarchies so that one can think of spart-tuples. Such an organization of sparts may correspond to flow ribbons or rakes used in flow visualization.

It should be obvious that using the appropriate sparts, spray rendering is capable of effects similar to those obtained by traditional surface, volume and flow visualization. Thus, by using this platform, a visualization user can effectively investigate the data set with a combination of different visualization techniques. The user also has the flexibility of mixing and matching different cans of sparts in visualizing, exploring, analyzing and hopefully gaining some insight on their data set. In addition, the spray paint metaphor provides a natural way of incorporating virtual environment interfaces within the same framework. Thus, one can envision scientists entering into a virtual world where they can pick different spray cans, walk around their data set, explore and highlight different features in the data, all within the realm of this virtual environment.

It should be noted that each spart has a changing set of local data and neighboring sparts. There is no specific requirements that the data to be rendered must lie within some grid system. Except for efficiency concerns, it does not matter whether the data are available at regular or irregular grid points and whether the data are dense or sparse. For sparse data sets, a spart may extend its local domain to a larger area, or it may simply not manifest itself if there is insufficient local data.

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. This is achieved by the following methods: (a) Adjust the nozzle of the spray can so that it has a wide area of coverage. After the initial spray, selectively highlight areas of interest with a narrower beam of sparts. (b) Adjust the density or the number of active sparts being sprayed. During the initial exploratory spray, use less but larger sparts. Using a combination of these two methods, one can progressively refine on earlier renderings and incrementally focus in on interesting sections of the data set.

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. Thus, sparts offer a modular and extensible mechanism for adapting to the changing needs of the user. It is expected that early designers for sparts will have to do programming. However, once a set of spart attributes has been created, a new spart can be generated interactively from a pick list or palette of targets and behaviors involving minimal or no programming. These attributes are described in the next section and are meant to give a glimpse of what a spart may contain.

3.2. The whole picture

Spray rendering provides a framework from which to apply diverse rendering techniques in a unified manner. It is born out of a need for scientists to be able to render large data sets with widely varying data structures without learning several different packages. The interface for launching sparts appears intuitive and encourages users to interactively explore their data sets.

There is no inherent restriction on the structure of the data. Data sets may be sparse or densely packed. They may be presented on regular grids or on curvilinear substrates. Data at each point may be scalar or vectors of arbitrary lengths. New data formats are easily incorporated by designing new sparts that handle them. Furthermore, there is no inherent restriction on the type of calculations that a spart can perform when searching for their target or how they will manifest and reproduce themselves.

With so few restrictions on the sparts it is valid to ask what distinguishes spray rendering from a collection of arbitrary processes which act upon a common data set. In essence, there is no difference. The utility of spray rendering comes from the user interface which allows different sparts to be selected in an ad-hoc manner. The ability to mix-and-match different sparts simultaneously provide the capability of producing images that are composed by using different visualization techniques.

Sparts make decisions based on their current state. Sparts have some notion of their current position and use local information to determine their next position or state. Another attribute of state is the current age of the spart, where age can be defined arbitrarily by the spart, but has been commonly understood to be the number of state changes since the spart was activated or born. At each state change throughout the lifetime of a spart, the spart can output an abstract visualization object (AVO) [Haber90], give birth to other sparts and/or die (see Figure 2). The AVO's resulting from spray rendering a data set may be thought of as visual data, for they can take on any intermediate form that is convenient for the target physical display device.

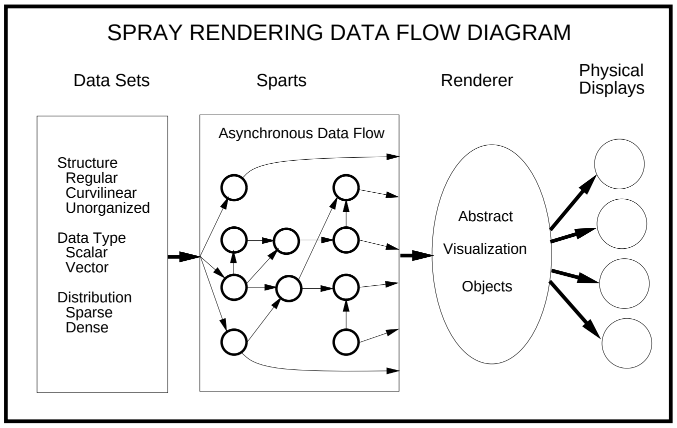

Figure 2. Arbitrary data types are processed by different sparts which leave invisible markers and output abstract visualization objects for the renderer to display.

The spray rendering user interface is both a window from which to view the AVO's output by sparts, both extant and extinct, and a platform from which to launch new sparts. However, it does not exercise control over the lives of sparts once they are launched. Each spart may be initiated as a separate process running on a distal machine, so long as all have access to a common database. The implementation of sparts may change to take advantage of the underlying architecture, be it parallel, vector or Von Neumann.

4. Sample sparts for spray rendering

This section outlines some examples of sparts that mimic current visualization techniques as well as some examples that go beyond current techniques. We start out by describing the canonical spart which can be used as a template for describing the major parts of a spart and a spray can.

SPRAY CAN:

State:

NozzleType(cone, flood, rake, etc.)	; pattern of spray
Location(x, y, z)	; initial nozzle location
Attitude(x, y, z)	; direction of spray
SpartType(SS, VP, FT, etc.)	; different type of sparts
SpartOrganization(single, tuple, hierarchical)	
SpartDistribution(n, d)	; number & distribution of sparts

Methods:

Spray()	; deliver a dose of sparts
LoadSpart(spart type)	; load new sparts in can
VaryDistribution(n, d)	; change density & distribution
VaryNozzle(nozzle type)	
MoveCan(x, y, z)	
PointCan(x, y, z)	

SPART:

State:

Position(x, y, z)	; current position of spart
Trajectory(x, y, z)	; current trajectory of spart
Age	; number of state changes since birth
Lifetime	; maximum state changes till death
SpartAppearance	; color, transparency, shape, ; size, texture, etc.
TraceAppearance	; color, transparency, shape, ; size, texture, etc.
DataType(point, surface, volume, etc.)	; data type managed by spart

Methods:

TargetFunction()	; determines if a target is reached ; *e.g.* surface, gradients, flows, etc.
DirectionFunction()	; calculates next trajectory
SpawnFunction()	; determines number of new sparts
DeathFunction()	; determines if spart should die
AVOFunction()	; outputs AVO's. *e.g.* spart color, ; object color, trace, etc.
MarkerFunction()	; outputs markers, see below
Map()	; modifies mapping, see below

The Map() function is used for mapping n-dimensional data sets without explicit spatial information, into a cartesian coordinate system that the spray cans and sparts understand. Up to three independent parameters can be mapped to the corresponding x,y,z world coordinates. The rest of the $n-3$ parameters are then treated as a vector of parameters at various world coordinates. In essence, this is a projection map of a higher dimensional space to three space. This usage of the Map() function allows the spray can to be positioned relative to the mapped 3D projection.

Another possible use of the Map() function is to calculate a set of new coordinates, different from the spray can coordinates, in which the spart will operate. Thus, it is possible to distinguish between the coordinate system where the spray can is being used and the coordinate system where the AVO's are rendered.

There are two types of data that a spart can output. The first are abstract visualization objects which are visual objects that show up on the physical display device. The second type of data output by a spart are markers. Markers are not visual objects, and are not rendered on the display. They are used by sparts to communicate with other cooperating sparts. In order to communicate, cooperating sparts are assumed to understand the marker format of the other sparts. All markers have a location that corresponds to the (x,y,z) location of the spart where they were dropped. They also have tags that identify what types of data are stored within the marker. The value and structure of the data that each tag identifies within a marker is determined by the spart that dropped it. Note that while markers have locations within the coordinate system of the original data, they do not modify the original data. They exist in a separate but equivalent coordinate system. No spart ever modifies the original data set.

4.1. Surface seeking sparts

A surface seeking (SS) spart travels forward until it intersects with a surface. The determination of what constitutes a surface is made locally by the spart. So, it can find more general surfaces than are present in the standard polygonal world. For example, a surface may be represented as a bilinear patch by a spart's four nearest points. Thus, these SS sparts can provide effects similar to iso-surface rendering and traditional polygonal rendering. The behavior of the spart is not limited to highlighting the entire polygonal surface that it hits. It may simply display the intersection point. Alternatively, the spart can blot the surface with a paint spot partially highlighting the area around the intersection point. Yet another possible behavior is for the spart to bounce off the intersected surface or continue through at an angle of refraction. While this seems like raytracing, the effects are different since the eye vector does not coincide with the spray nozzle vector. A closer counterpart to this would be a step in a progressive radiosity algorithm where each patch has to determine where to shoot its accumulated energy.

Figures 3 and 4 show two different behaviors for a SS spart when applied to a terrain that is modeled as a height field. The former highlights the entire polygonal surface that is intersected while the latter highlights only a small neighborhood around the intersection point.

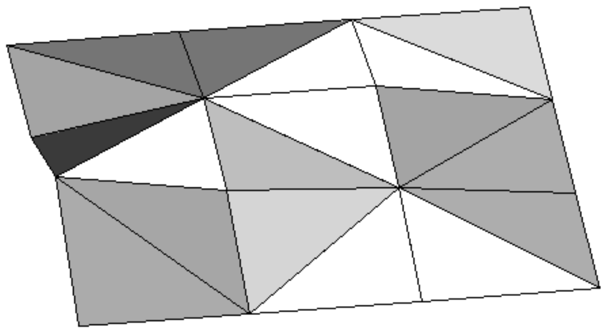

Figure 3. The entire polygon is highlighted as soon as a spart hits it.

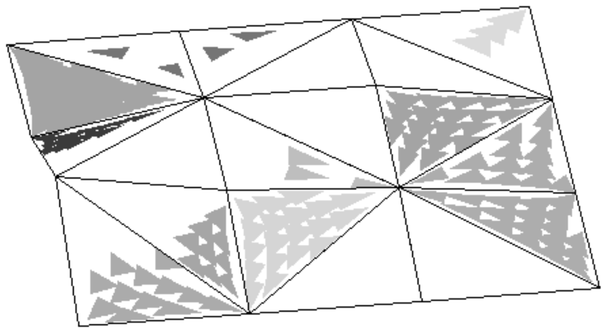

Figure 4. A small paint spot (splat) is displayed where a spart hits a surface. Polygon boundaries are displayed for illustration purposes only and are not part of the spart behavior.

4.2. Volume penetrating sparts

The volume penetrating (VP) sparts do not seek out surfaces. In fact, they do not have specific targets. Instead, they act like high energy particles that are bombarding the data sets. The visual effects of passing these sparts through the data set depend on their behavioral description. For example, ray marching algorithms can be simulated with VP sparts that accumulate density and integrate color information along its path. If the sampling ray is required to diverge into many rays, the VP spart can likewise spawn off additional VP sparts. If an average of samples is required, then the VP spart can leave its contribution to the final value as a marker. Each subsequent VP spart that encounters the same target can make the appropriate adjustment to the marker.

Another behavioral manifestation of VP sparts would be to imagine a slightly less energetic spart or a data set that exerts very strong influence on the VP spart so as to influence its path. This type of spart may be used to visualize the refraction effects of light particles going through materials of varying density. Even if the data is very sparse, VP sparts can be used to visualize how light bends as it is influenced by strong gravitational forces.

4.3. Flow tracking sparts

Flow tracking (FT) sparts are ideal for visualizing vector fields. The FT 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 surface or accumulated energy. Therefore, FT sparts manifests themselves by leaving a trace of their path as they advance from one state to another. FT sparts may work in pairs or groups so as to form flow ribbons and rakes respectively.

4.4. 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 (GC) spart that simply removes the visual cues from the rendered image. Typical uses for such a GC 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 note that the original data set is left untouched by this and all other types of sparts.

4.5. Other eccentric sparts

The power and richness of spray rendering comes from the ability of specifying arbitrary attributes for the sparts. Novel visualization effects can be created by defining the appropriate spart target and behavioral attributes. Furthermore, sparts can highlight relationships among different data parameters. Below is a collection of some possible sparts that come to mind.

Define a SS spart whose behavior is to paint the surface according to its height, slope, temperature or other variables. Once a surface is found, make arrangements for the spart to slide down the slope of the surface thereby simulating the path that a rain drop would make as it rolls down the slope because of gravity. If FT sparts were presented with a scalar field, rather than a vector field, they can be used to identify iso-potential fields by highlighting paths or surfaces the sparts trace out as they are forced through the field. For example, contour lines and surfaces showing isobars are formed by sparts that travel through a section of data where different data points are exerting an equal amount of pressure (or whatever the relevant parameter happens to be). Additionally, aging and interaction among parameters can be highlighted by sparts that may represent oil spill particles being dispersed by wind, current and tidal forces as well as being broken down by chemical reactions. Sparts with arbitrarily complex shapes such as leaves and perhaps birds or fishes can be used. It is also possible to compare data sets with historical data and where the variance exceeds a certain tolerance, those points may be emphasized. That is, the spart attributes are combined with data pre-processing and feature extraction operations. In short, the combinations are endless.

5. Summary

The immediate benefit of spray rendering is the ability to combine, in a single image, the abstract visualization objects which have been produced by any combination of sparts. Thus, by learning a single visualization system, the scientist can produce images employing multiple resolutions and multiple rendering techniques. In addition, since the spart designer has the freedom of defining new targets and behaviors in their sparts, it is relatively easy to produce novel visualization effects.

Another benefit of spray rendering is the incremental manner in which an image can be rendered. With spray cans, various sparts can be tested on a subset of the data to determine the effect of applying different types of sparts. The user selects the type of sparts to fill up the spray can, and also chooses various spray nozzle properties. For instance, the shape of the nozzle can be circular which delivers a cone of sparts in a directed manner to a particular region of the data set. Alternatively, the nozzle can be so wide that it floods the entire screen with sparts. This allows interactive exploration of very large data sets.

Finally, spray rendering allows for a modular way of combining rendering algorithms on the fly, without programming. The spray rendering user interface allows for the construction of sparts from a palette of pre-wired behaviors. A spart may be constructed by selecting target features which drive the movement of the spart within the data, and behaviors to control how the spart manifests itself and the target feature. Other parameters that represent the lifetime, shape and color of the spart can also be specified in a similar manner. Thus, with a very small set of predefined attributes, the user is able to create a large number of unique sparts. To draw on the analogy of paint particles, this is akin to mixing paints to come up with just the right color.

We have shown that spray rendering has great potentials. At the same time, it must address several issues of practical concern. With so much flexibility built into the system, the challenge is to design a system which is reasonably efficient. Several factors affect the interactivity of sparts and its apparent performance. The most important factor is the number of active sparts that must be tracked. The complexity of the spart geometry, its target and behavioral attributes are also primary factors that influence the performance of sparts. Another significant factor is the size of a spart's local neighborhood. As the spart traverses through the data, it's local set of data points must be updated to include new ones that come within its spatial domain and discard those that are too far away. Obviously, judicious tradeoffs need to be made between the amount and the complexity of sparts, between interactivity and visual resolution, and between the amount of particle to particle cooperation and the cost of communication among sparts. Fortunately, the nature of sparts make them very suitable for parallel implementation. However, care must be taken to ensure that implementation details, such as taking advantage of any underlying parallel or vector processors, are transparent to the users' view of the virtual cans of spray paint.

6. References

[Brys92] Bryson, S. and C. Levit, ''The Virtual Wind Tunnel,'' *IEEE Computer Graphics and Applications* **12**(4), pp. 25-34 (1992).

[Chen85] Chen, L. S., G. T. Herman, R. A. Reynolds, and J. K. Udupa, ''Surface Shading in the Cuberille Environment,'' *IEEE Computer Graphics and Applications* **5**(12), pp. 33-43 (1985).

[Craw92] Crawfis, R. and N. Max, ''Direct Volume Visualization of Three-Dimensional Vector Fields,'' *ACM Workshop on Volume Visualization*, pp. 55-60 (1992).

[Dreb88] Drebin, R., L. Carpenter, and P. Hanrahan, ''Volume Rendering,'' *Computer Graphics* **22**(4), pp. 65-74 (1988).

[Ells88] Ellson, R. and D. Cox, ''Visualization of Injection Molding,'' *Simulation* **51**(5), pp. 184-188 (1988).

[Fuch77] Fuchs, H., Z. M. Kedem, and S. P. Uselton, ''Optimal Surface Reconstruction from Planar Contours,'' *Communications of the ACM* **20**(10), pp. 693-702 (1977).

[Gold85] Goldwasser, S. M., R. A. Reynolds, T. Bapty, D. Baraff, J. Summers, D. A. Talton, and E. Walsh, ''Physician's Workstation with Real-Time Performance,'' *IEEE Computer Graphics and Applications* **5**(12), pp. 44-56 (1985).

[Habe90] Haber, R. B. and David A. McNabb, ‘‘Visualization Idioms: A Conceptual Model for Scientific Visualization Systems,’’ pp. 74-93. in *Visualization in Scientific Computing*, ed. G. M. Nielson, B. Shriver and L. J. Rosenblum, IEEE Computer Society Press (1990).

[Helm91] Helman, J. L. and L. Hesselink, ‘‘Visualizing Vector Field Topology in Fluid Flows,’’ *IEEE Computer Graphics and Applications* **11**(3), pp. 36-46 (1991).

[Hopp92] Hoppe, H., T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, ‘‘Surface Reconstruction from Unorganized Points,’’ *Computer Graphics* **26**(2), pp. 71-78 (1992).

[Hult90] Hultquist, J., ‘‘Interactive Numerical Flow Visualization Using Stream Surfaces,’’ Tech. Rep. RNR-90-009, NASA Ames Research Center, (1990).

[Kauf88] Kaufman, A. and R. Bakalash, ‘‘Memory and Processing Architecture for 3D Voxel-Based Imagery,’’ *IEEE Computer Graphics and Applications* **8**(11), pp. 10-23 (1988).

[Laur91] Laur, D. and P. Hanrahan, ‘‘Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering,’’ *Computer Graphics* **25**(4), pp. 285-288 (1991).

[Levo88] Levoy, M., ‘‘Display of Surfaces From Volume Data,’’ *IEEE Computer Graphics and Applications* **8**(5), pp. 29-37 (1988).

[Lore87] Lorensen, W. E. and H. E. Cline, ‘‘Marching Cubes: A High-Resolution 3D Surface Construction Algorithm,’’ *Computer Graphics‘* **21**(4), pp. 163-169 (1987).

[McCo87] McCormick, B. H., T. A. DeFanti, and M. D. Brown, ‘‘Visualization in Scientific Computing,’’ *Computer Graphics* **21**(6) (1987).

[Phil90] Phillips, R. L., B. Cabral, C. L. Hunter, R. B. Haber, G. V. Bancroft, T. Plessel, F. Merritt, P. P. Walatka, and L. J. Rosenblum, ‘‘Scientific Visualization at Research Laboratories,’’ pp. 209-253 in *Visualization in Scientific Computing*, ed. G. M. Nielson, B. Shriver and L. J. Rosenblum, IEEE Computer Society Press (1990).

[Reev83] Reeves, W. T., ‘‘Particle Systems − A Technique for Modelling a Class of Fuzzy Objects,’’ *Computer Graphics* **17**(3), pp. 359-376 (1983).

[Reyn87] Reynolds, C. W., ‘‘Flocks, Herds, and Schools: A Distributed Behavioral Model,’’ *Computer Graphics* **21**(4), pp. 25-34 (1987).

[Rhod85] Rhodes, M. L., Yu-Ming Azzawi, E. Tivattanasuk, A. Pang, K. Ly, H. Panicker, and R. Amador, ‘‘Curved-Surface Digital Image Reformations in Computed Tomography,’’ *Proceedings of SPIE, Medical Image Processing* **593**, pp. 89-95 (1985).

[Sabe88] Sabella, P., ‘‘A Rendering Algorithm for Visualizing 3D Scalar Fields,’’ *Computer Graphics* **22**(4), pp. 51-55 (1988).

[Sims90] Sims, K., ‘‘Particle Animation and Rendering Using Data Parallel Computation,’’ *Computer Graphics* **24**(4), pp. 405-413 (1990).

[Szel92] Szeliski, R. and D. Tonnesen, ‘‘Surface Modeling with Oriented Particle Systems,’’ *Computer Graphics* **26**(2), pp. 185-194 (1992).

[Upso88] Upson, C. and M. Keeler, ‘‘V-Buffer: Visible Volume Rendering,’’ *Computer Graphics* **22**(4), pp. 59-64 (1988).

[VanG92] Van Gelder, A. and J. Wilhelms, ‘‘Interactive Animated Visualization of Flow Fields,’’ *ACM Workshop on Volume Visualization*, pp. 47-54 (1992).

[Wejc91] Wejchert, J. and D. Haumann, ‘‘Animation Aerodynamics,’’ *Computer Graphics* **25**(4), pp. 19-22 (1991).

[Wilh90] Wilhelms, J., J. Challinger, N. Alper, S. Ramamoorthy, and A. Vaziri, ‘‘Direct Volume Rendering of Curvilinear Volumes,’’ *Computer Graphics* **24**(5), pp. 41-47 (1990).