Spray Rendering: Visualization Using Smart Particles

Alex Pang and Kyle Smith
Board of Studies in Computer and Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064

Abstract

We propose a new framework for doing scientific visualization. The basis for this framework is a combination of particle systems and behavioral animation. Here, particles are not only affected by the field that they are in, but can also exhibit different programmed behaviors. An intuitive delivery system, based on virtual cans of spray paint, is also described to introduce the smart particles into the data set. Hence the name spray rendering. Using this metaphor, different types of spray paint are used to highlight different features in the data set. Spray rendering 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; (4) it is modular, extensible and provides scientists with the flexibility for exploring relationships in their data sets in natural and artistic ways.

1 Introduction

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, one can imagine a formula X spray paint which uses data values rather than the color of the paint to highlight specific parameters. This might be similar to wood stain which highlights wood grain and texture. One can also imagine a formula Y spray paint that can penetrate through data volumes just as x-ray particles can penetrate through bodies. Furthermore, one can create a formula Z spray paint that will be activated only when it encounters a certain *range* of data values or a combination of different parameters. In a way, these particles are performing a rudimentary form of feature extraction by seeking out target features of interest. Once the 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.

2 Challenges to visualization

In recent years, there has been significant advances in visualization techniques. Among these are work in flow visualization with particles [7, 6, 16], direct volume visualization [3, 15, 14, 9, 2] and surface visualization [1, 11, 8, 10]. In addition, most of these techniques are now made accessible to common users through visualization platforms such as AVS, Khoros and Explorer which incorporate very friendly user interfaces.

Despite these advances, there are still some major obstacles to be addressed. For example, most of the

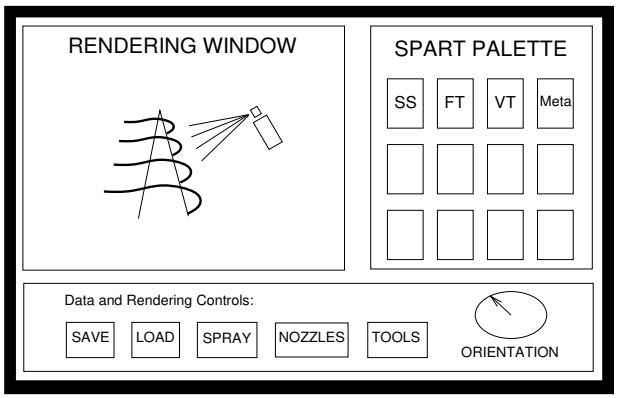

Figure 1: A mock panel showing a shelf of sparts that a user can choose from to fill a spray can. Data are then selectively visualized as the user move the virtual spray can around the data set.

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. 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. The last requirement places strong demands on the computational power, communication bandwidth, intelligent hierarchical organizations and requires difficult trade-off decisions.

3 Spray rendering

3.1 Design goals

Some of the concerns above are addressed in the design goals of spray rendering. The spray rendering system should be modular and extensible so that it can grow with the needs of the users. It should be able to handle sparse and dense, regularly and irregularly gridded data. It should also support interactive visualization through selective progressive refinement. And finally, it should provide a unified framework for both users and developers to create and implement visualization products and algorithms.

3.2 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 and particle systems in that each particle is endowed with instructions on what to seek out and how to react to its changing environment. Particle systems [13] were originally developed as a means of modeling natural phenomena such as fire, grass and forests. Recently, particle systems have found their way to fluid flow visualization. Behavioral animation [12], 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. More recently, interaction of the characters with their environment has been incorporated. An example of this is the leaves in the wind simulation [17]. 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 [4] or 3D icons. For instance, sparts may be shaped like leaves. These particles can then be sprayed or thrown into the wind to highlight wind flow and the presence of vortices. Secondly, sparts have a set of programmed targets that instruct them to seek out features in their local neighborhoods. A sample target is to 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). A third attribute of sparts is their programmed 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 by having the paths of the sparts be 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.

By using the appropriate sparts, the visualization user can effectively investigate the data set with a combination of different visualization techniques similar to those obtained by traditional surface, volume and flow visualization. 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. The spray paint metaphor also 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 requirement that the data to be rendered must lie within some grid system. Data at each point may be scalar or vectors of arbitrary lengths. 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: (a) adjusting the nozzle of the spray can so that it has a wide area of coverage. After the initial spray, highlight areas of interest selectively with a narrower beam of sparts. (b) adjusting 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 targets and behaviors have been created, a new spart can be generated interactively by mixing and matching targets and behaviors with minimal or no programming.

4 System components

There are several components that make spray rendering functional. First of all, there is the concept of a spray can for launching or delivering the sparts to the relevant region in the data set. Secondly, there is the concept of filling the spray can with sparts that exhibit different behaviors. Finally, there is the concept that a spart is a dynamic collection of targets and behaviors which can be mixed and matched to produce sparts with different effects. Beneath these concepts is a flow diagram (see Figure 2) which shows how sparts act upon data sets, either individually or in cooperation with other sparts, to produce abstract visualization objects (AVO) [5] for the renderer to display.

4.1 Virtual spray cans

At any time, multiple spray cans may be filled and ready to fire. Each can would therefore need an identifier to distinguish itself. In addition, each can has the following definitions:

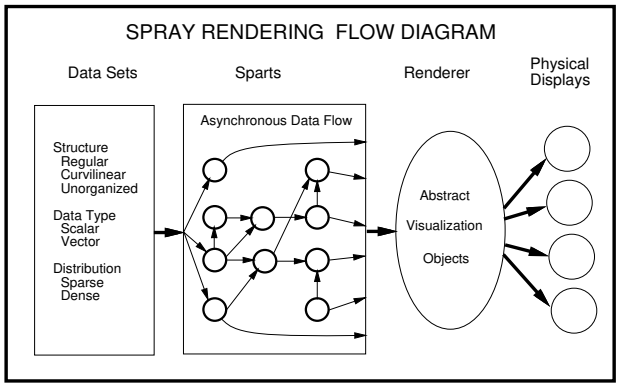

Figure 2: Arbitrary data types are processed by different sparts which leave invisible markers and output abstract visualization objects for the renderer to display.

State:
NozzleType(cone, flood, rake, ...);
pattern of spray
Location(x, y, z);
initial nozzle location
Attitude(x, y, z);
direction of spray
SpartType(SS, VP, FT, ...);
different type of sparts
SpartDistribution(n, d);
number and distribution of sparts
Methods:
Spray();
deliver a dose of sparts
LoadSpart(spart type);
load new sparts in can
VaryDistribution(n, d);
change density and distribution
VaryNozzle(nozzle type)
change nozzle type
MoveCan(x, y, z)
reposition can
PointCan(x, y, z)
change direction of spray

4.2 Smart particles

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 state attribute 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 AVO or leave an invisible marker, give birth to other sparts and/or die. The AVOs 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.

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
Behaviors;
list of targets and behaviors
DataType(point, surface, volume, ...);
data type managed by spart
Methods:
AddBehavior();
adds a new behavior to the spart
DeleteBehavior();
deletes a behavior from the spart
SpawnFunction();
determines number of new sparts
DeathFunction();
determines if spart should die

4.3 Targets and behaviors

The careful reader will notice that the spart state contains a list of behaviors which contains both targets and behaviors. This is because we can generalize targets as behaviors. The main distinction between the two is that behaviors may output AVOs for subsequent rendering while targets may leave invisible markers for use by other sparts or behaviors. From the perspective of a spart, these are the two types of data that it can output. Markers are used by sparts to communicate with other cooperating sparts. They may also be used within the same spart for behaviors to communicate with each other. In order to communicate, 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 identi-

fies 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.

State:
SpartAppearance;
color, transparency, shape, size, ...
Methods:
MovePosition();
moves spart to a new position
ChangeTrajectory();
modifies spart's trajectory
GetData();
obtains data of local neighborhood
PutAVO();
outputs AVOs. e.g. *color, trace, ...*
PutMarker();
leaves invisible marker

5 Sample sparts for spray rendering

5.1 Surface seeking sparts

A surface seeking (SS) spart travels forward until it intersects 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 based on 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.

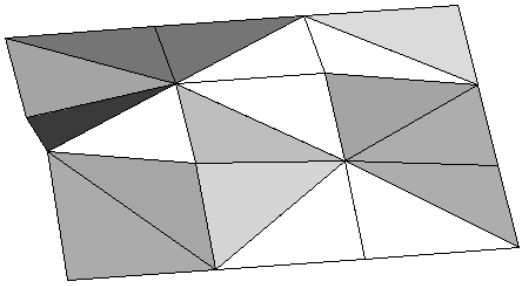

Figure 3: The entire polygon is highlighted as soon as a spart hits it.

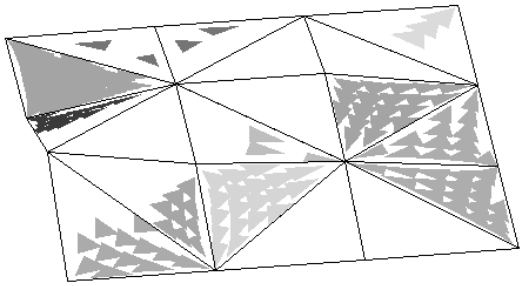

Figure 4: A small paint spot (triangular splat) is displayed where a spart hits a surface. Polygon boundaries are displayed for illustration purposes only and are not part of the spart behavior.

5.2 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 set. The visual effect 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 their 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.

5.3 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.

5.4 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 and cleaning the rendered scene to reduce 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.

5.5 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.

6 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. Favorite combinations of targets and behaviors can also be saved as a new spart. Another benefit of spray rendering is the incremental manner in which an image can be rendered. Combining the intuitive and selective mechanisms of a spray can and the adjustable granularity of a spart's local neighborhood allows one to interactively explore very large data sets. Finally, spray rendering allows for a modular way of combining rendering algorithms on the fly, without programming. Note that each type of spart is in effect a different visualization technique. New types of spart can be constructed by simply combining different targets and behaviors. Thus, with a very small set of predefined targets and behaviors, the user is able to create a large number of unique sparts.

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, its 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.

Acknowledgements

We wish to thank Naim Alper for implementing some of the earlier form of sparts among which are iso-sparts, ribbon-sparts and ray-sparts. Support for this work is funded in part by the Office of Naval Research grant no. N-00014-92-J-1807. Questions regarding the article can be directed to pang@cse.ucsc.edu.

References

[1] L. S. Chen *et al.*, "Surface Shading in the Cuberille Environment," *IEEE Computer Graphics and Applications*, Vol. 5, No. 12, pp. 33-43, 1985.

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

[3] R. Drebin, L. Carpenter and P. Hanrahan, "Volume Rendering," *Computer Graphics*, Vol. 22, No. 4, pp. 65-74, 1988.

[4] R. Ellson and D. Cox, "Visualization of Injection Molding," *Simulation*, Vol. 51, No. 5, pp. 184-188, 1988.

[5] R. B. Haber and D. A. McNabb, "Visualization Idioms: A Conceptual Model for Scientific Visualization Systems," in *Visualization in Scientific Computing*, edited by G. M. Nielson, B. Shriver and L. J. Rosenblum, IEEE Computer Society Press, pp. 74-93, 1990.

[6] J. L. Helman and L. Hesselink, "Visualizing Vector Field Topology in Fluid Flows," *IEEE Computer Graphics and Applications*, Vol. 11, No. 3, pp. 36-46, 1991.

[7] J. Hultquist, "Interactive Numerical Flow Visualization Using Stream Surfaces," *NASA Ames Technical Report*, RNR-90-009, 1990.

[8] A. Kaufman and R. Bakalash, "Memory and Processing Architecture for 3D Voxel-Based Imagery," *IEEE Computer Graphics and Applications*, Vol. 8, No. 11, pp. 10-23, 1988.

[9] D. Laur and P. Hanrahan, "Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering," *Computer Graphics*, Vol. 25, No. 4, pp. 285-288, 1991.

[10] M. Levoy, "Display of Surfaces From Volume Data," *IEEE Computer Graphics and Applications* Vol. 8, No. 5, pp. 29-37, 1988.

[11] W. E. Lorensen and H. E. Cline, "Marching Cubes: A High-Resolution 3D Surface Construction Algorithm," *Computer Graphics*, Vol. 21, No. 4, pp. 163-169, 1987.

[12] W. T. Reeves, "Particle Systems - A Technique for Modelling a Class of Fuzzy Objects," *Computer Graphics*, Vol. 17, No. 3, pp. 359-376, 1983.

[13] C. W. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model," *Computer Graphics*, Vol. 21, No. 4, pp. 25-34, 1987.

[14] P. Sabella, "A Rendering Algorithm for Visualizing 3D Scalar Fields," *Computer Graphics*, Vol. 22, No. 4, pp. 51-55, 1988.

[15] C. Upson and M. Keeler, "V-Buffer: Visible Volume Rendering," *Computer Graphics*, Vol. 22, No. 4, pp. 59-64, 1988.

[16] J. J. van Wijk, "Rendering Surface-Particles," *Visualization'92*, pp. 54-61, 1992.

[17] J. Wejchert and D. Haumann, "Animation Aerodynamics," *Computer Graphics*, Vol. 25, No. 4, pp. 19-22, 1991.