Spray Rendering as a Modular Visualization Environment

Alex Pang and Craig Wittenbrink
Baskin Center for Computer Engineering & Information Sciences
University of California, Santa Cruz
Santa Cruz, CA 95064

Spray rendering is a modular visualization environment (MVE) similar in many ways to AVS, IBM DX, and SGI Explorer. It retains the main attracting features such as modularity, extensibility and relative ease of use. Its distinguishing features are its execution flow and the finer granularity of visualization modules. In this paper we briefly describe the *Spray* rendering environment together with its Mix & Match capabilities and its collaborative visualization extensions. We then highlight its differences from other systems and list some of the advantages offered by *Spray* rendering. Finally, we look at some of the open issues and challenges facing our system and MVEs in general.

Spray Rendering

Spray rendering uses the metaphor of spray cans to paint and visualize data sets. Conceptually, visualization users grab and aim spray cans into their data sets. Depending on the type of paint in the can, different data features are highlighted and rendered. The paint particles are referred to as sparts which stands for smart particles. They combine the power of both particle systems [Ree83] and behavioral animation [Rey87] to seek out and highlight features of interest in the data set. Sparts can be programmed to produce effects similar to other popular visualization algorithms. Color plates 1-5 illustrate some of the visualizations produced by different types of sparts. For example, a surface seeking spart finds isosurfaces just like the marching cubes algorithm. Other sparts include flow trackers for visualizing vector fields, and x-ray sparts for a quick and dirty inspection of scalar fields. More exotic sparts may be made, for example a spart that lays down arrow icons in a vector field only in areas where the pressure is in a given range.

Operationally, each spray sends out a number of sparts into a local region in the data set. Each spart then travels through the data set according to its programmed position update function, looking for its programmed target functions, depositing or exhibiting appropriate visual actions wherever targets are found, and checking whether it should terminate itself or spawn new sparts. Note that the programmed behavior of a spart is independent of how they are sent to the data. Thus other possible user interactions, some of which have also been implemented, include flooding the data set, flashlights to highlight the visual effects which disappear when the can is pointed away, sprinklers for continuous or intermittent sprays, probes which report values in some local data region, and event driven sparts which react to changes in the data stream.

Mix & Match

The type of sparts determine the visualization produced. Sparts are either pre-defined or constructed interactively. The Mix & Match [PA94] feature of *Spray* rendering provides both a textual

and graphical interface for building sparts from components. Components can be classified as one of four categories: targets, behaviors, position updates, and spawn/death functions. Each component is implemented as a function. These can then be combined to specify a particular spart. For example, a surface seeking spart can be defined by a target function that looks for a threshold value in the spart's vicinity, leaves a surface when the threshold target is found, moves some distance along its sprayed trajectory, and decides to terminate itself once it is out of the data space. This form of spart construction provides the modularity and code reuse of many spart components in creating different sparts. It also promotes extensibility since each component is small and easy to write.

CSpray

CSpray is Collaborative *Spray* rendering [PWG95], and includes the necessary constructs, interface mechanisms, and network programming for many distributed users to work at once in a single workspace. The main research issues for *CSpray* are the use of visualization-primitive streams, instead of video, or raster streams, to communicate, the efficient use of available network bandwidth, the appropriate metaphors for distributed, scalable, and fair interaction, and the inclusion of varying grades of client service. With visualization-primitive streams, the possibilities for compression are greater, and participants have more flexibility depending on their rendering power. We are investigating visualization stream creation as a method of distributed system resource allocation with graphics workstations, from SGI Reality engines to X terminals. Using different grades of service, we have created a collaborative environment, where the resources available through the network give all users access to the session. Work is just beginning on collaboration over an ATM (Asynchronous Transfer Mode) network.

For more information and details on *Spray*, Mix&Match, *CSpray*, and our ongoing work on environmental visualization please refer to the visualization section under the REINAS project heading in World Wide Web–Uniform Resource Locator, `http://www.cse.ucsc.edu/research/projects`, and to our papers [PWG95, PA94, Pan94].

Similarities and Differences

Spray rendering shares several prominent features with commercial MVEs such as: the modularity of the data and visualization transformation blocks, the extensibility of the system by adding new modules, the adaptability to distributed and parallel implementations, and the relative ease of use especially with the visual programming interface for creating visualization networks.

There are two main difference between *Spray* and commercial MVEs: *Granularity*: The granularity of spart components are finer than the modules in commercial MVEs in two senses: the components are simpler and smaller, and they are programmed to process a subset of the data. Spart components are easier to write because they generally have very specific and simple tasks such as: advance one step in the spart's current direction, generate a polygon at the spart's current location, find an isovalue in the spart's vicinity, or perform compound boolean (and/or/not) operations on input values. Since spart components have a finer granularity, it also offers more flexibility in terms of combining them to generate new sparts. Additionally, the granularity of spart components calls for traversing input data streams and creating visualization primitives rather than images, or entire scenes or objects. This allows massive parallelism, high efficiency with enormous data sets, and fine enough threading to allow for many possibilities in the scheduling and multitasking of executing spart processes.

Execution flow: Instead of moving data through modules in a data flow fashion, the execution flow in *Spray* rendering moves modules through the data. Conceptually, one can think of sending the sparts as intelligent agents to the data space to look for targets and highlight features of interest. As a consequence of this, we had to analyze and rethink a lot of the visualization algorithms and cast them in a form that would work on a particular subset of the data at a particular time. The process is not unlike converting sequential code to take advantage of the data parallel programming paradigm. In the case of visualization methods, they can be generalized to having two basic operations: a feature or target extraction process and a "make the feature visible" process. These ideas are encapsulated in the composition of a spart. The spart execution flow imposes a simple structure on a spart's life cycle – a spart simply goes through its list of target functions and executes its behavior functions whenever/wherever the targets are satisfied; it then changes its position using its position update function and decides whether it should terminate or spawn new copies of itself. Note that all spart components need not be present. For example, target functions are usually absent in sparts that operate on vector fields since these sparts are normally carried around by the flow. The missing components are simply skipped during the execution of a spart.

Challenges

Kaufman and Nielson [NK94] use the metaphor of life to place the visualization field at a stage where it has just finished high school and is about to go to college. MVEs cannot out-pace the basic visualization tools that they incorporate, so more basic research in visualization is needed. In terms of the integration, usability, and value added features, MVEs share the concept of modularity and extensibility. They are also easy to use, and the ability to add and share independent modules contribute to the growing number of MVE users. If MVEs today are also about to embark on collegiate challenges, then we think two major challenges are: Integration of visualization and database and effective use of parallel and distributed resources for rendering.

Today, data sets are mostly stored in files, loaded, then visualized. One problem with this approach is that one must know the format of the file. Special routines or procedures must be created before the data can even be loaded. Notes must also be kept on which routines were used to read the file so that work is not duplicated when the same file needs to be read again sometime later. What is needed is a push away from files and towards better integration between visualization and database components. We need to take advantage of database technology to visualize large data sets interactively, and push the data formatting problem to a data load path issue. The latter will also encourage sharing of data instead of just MVE modules. Perhaps a common application programming interface (API) that supports query access, (geographic, spatial, temporal, and value) clipping, resampling, and conversion will help push MVEs into their first year of college.

Parallel and distributed rendering allow greater scaling in the amount of rendering requests that can be satisfied. With queries creating demands for terabytes of data, there can also be large demands on the rendering required for a given data set. Many MVEs batch their requests, and it's fairly easy to overload a single workstation. Of course it's easy to say that parallel and distributed rendering can markedly improve performance for interactive rendering, but it has proved challenging to do so. Transparent, effective use of remote parallel and distributed resources for rendering is a challenge for all MVE developers.

In *Spray* rendering, we also face these challenges. Specifically, we are working on the parallelization of sparts, handling irregular grids and scattered points, and better integration with databases. To a certain extent, the database issue is the crux of the challenge. That is, given a database that can support queries of spatial and temporal data relationships at a fine enough and

fast enough level, each spart can make their own query as they travel through data space. The problem of dealing with a large number of scattered points is made simpler as the number of points is much smaller and only within the immediate vicinity of the spart. But, with currently available databases' performance, we are still forced to solve much of the problem in querying, caching, and traversing data in the visualization software itself, leaving us with significant challenges to graduate to the next level.

References

[NK94] Gregory M. Nielson and Arie E. Kaufman. Guest editor's introduction: Visualization graduates. *IEEE Computer Graphics and Applications*, 14(5):17 – 18, 1994.

[PA94] Alex Pang and Naim Alper. Mix & Match: A construction kit for visualization. In *Proceedings: Visualization '94*, pages 302 – 309. IEEE Computer Society, 1994.

[Pan94] A. Pang. Spray rendering. *IEEE Computer Graphics and Applications*, 14(5):57 – 63, 1994.

[PWG95] Alex Pang, Craig M. Wittenbrink, and Tom Goodman. CSpray: A collaborative scientific visualization application. In *Proceedings SPIE IS &T's Conference Proceedings on Electronic Imaging: Multimedia Computing and Networking*, to appear 1995.

[Ree83] W. T. Reeves. Particle systems: A technique for modeling a class of fuzzy objects. *Computer Graphics*, 17(3):359 – 376, 1983.

[Rey87] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *Computer Graphics*, 21(4):25 – 34, 1987.

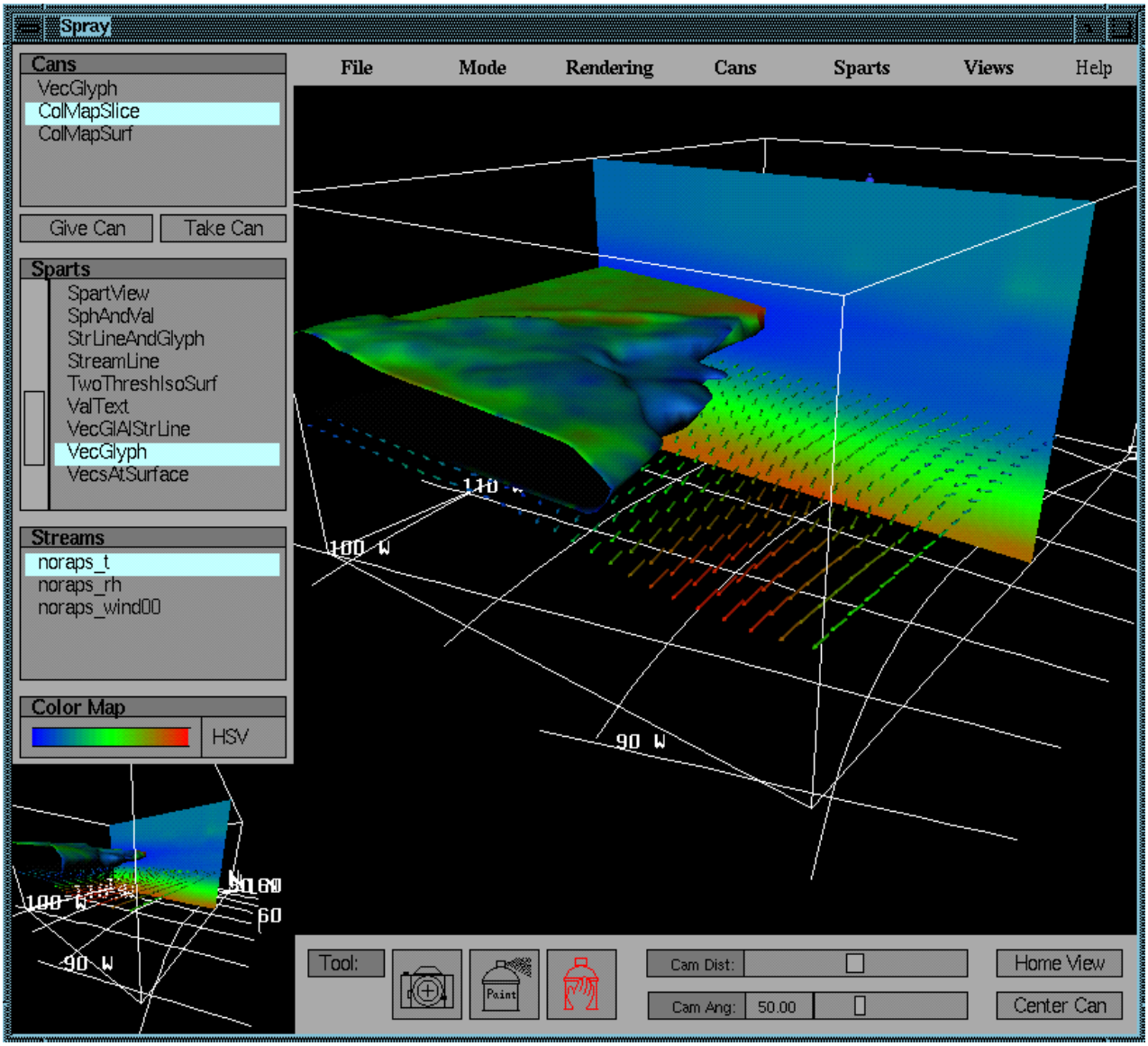

Figure 1: *Spray* rendering interface. Visualization shows temperature inversion isosurface colored with humidity values, vector glyphs to represent the wind field, and a pseudo-colored cross sectional slice of the temperature field.