Journal for Geometry and Graphics
Volume 3 (1999), No. 1, 47–55

SiPaViS — A Toolkit for Scientific Visualization and Simulation

Fernando P. Birra, Manuel J. Próspero

Computer Science Department, New University of Lisbon
P-2825 Monte Caparica, Portugal
emails: fpb@di.fct.unl.pt, ps@di.fct.unl.pt

Abstract. This paper introduces a new approach for scientific visualization and simulation based on a particle system. The main characteristics of the toolkit we designed and implemented at the New University of Lisbon are presented. Along with particles, fields, and particle sources, the central concept of the framework introduces the interaction as the fourth primary class of objects. The result is that a large diversity of techniques for data visualization and simulation can easily be integrated. A user of the toolkit can write a relatively short script in order to build a specific application program. This script is interpreted by SiPaViS (the name of this toolkit) according to the rules of a given grammar and on the basis of the existent re-usable classes. However, a programmer can always extend the set of predefined classes by writing the corresponding code in a common programming language. Some examples are given in the paper under the scope of application areas such as volume rendering through ray-casting implementation and also visualization of vector fields.

Key Words: data visualization, particle system

MSC 1994: 68U05

1. Introduction

Most commercial products offering easy to use and upgradable data visualization environments are built around the dataflow model. The user draws a network of modules that transform the data as it flows from module to module sequentially.

Yet there are some drawbacks associated with these systems. Firstly, the system performance degrades very quickly as data sets increase in size, since at least one module must process all the data. Secondly, the data types supported internally are sometimes too restrictive. And finally, the physical support of the data is generally a file, not allowing a steering mode control in the simulation process whenever the user interpretation from the visualization results may drive the execution of the simulation itself.

ISSN 1433-8157/$ 2.50

The primary goal of SiPaViS is to provide a framework to support different scientific visualization techniques based on a particle system. The use of particles has some relevant benefits [4]. For instance, they can work as probes sent into the environment to seek and extract the desired attributes at a very precise location or subspace. Examples of this metaphor include vector field visualizations or direct volume rendering.

But we also might want to simulate several kinds of interactions between objects. Our approach is to regard particles as self-contained entities that can react to each other. This alternative allows the simulation on a basis of behaviors programmed on top of SiPaViS. Behavioral animation [7] was then selected as a case study: the simulation of artificial fishes being able to swim in a school and to avoid obstacles was fully implemented [1].

2. Main Concepts

The conceptual model behind SiPaViS consists of four types of entities: particles, fields, particle sources and interactions. Any algorithm or simulation should be accomplished by creating and connecting different objects belonging to those four types.

We adopted an object-oriented approach to build the toolkit, with each type of object represented as an abstract class. The system is programmed at two distinct levels. At the programmer level, new classes of objects are derived directly or indirectly from the abstract classes while, at the user level, objects from existent classes are created and the connections between them are established using a script to be interpreted by the system.

Particles are represented as moving points in 3D space. The simplest particle in the toolkit has the following attributes: mass, position, speed, accumulated forces and age.

Field objects are available with the purpose of defining the surrounding properties of the environment where particles move. Two major types of fields can be implemented in SiPaViS. Analytically defined fields are easy to implement and can be inquired about the corresponding value at any point in 3D space. However, the most common fields found in scientific visualization are discrete, since they result from an intrinsic sampling process. These fields' values are usually stored in a file and an interpolation scheme is then provided to estimate values for other intermediate positions.

During each time step, a field may be asked for evaluation several times as a result of interactions currently defined. In SiPaViS, interactions have a new and important role. Since there is no direct communication between particles and fields or between groups of particles, those interactions are used as objects to set the new attribute values for particles by making them interact with each other and with the surrounding environment defined by fields.

The domain of an interaction is a list of sets. Each set may be composed, either by a single particle or field object, or by all objects of a specified subclass of particles or fields. Interactions are applied to each tuple extracted from the Cartesian product of all domain sets. The semantics of interactions (i.e., what kind of particle attributes they change and how these changes are computed) is completely left to the programmer as can be seen in Fig. 1, using the Objective C programming language [5].

Particle sources are needed to dynamically create particles during a simulation. These objects control when, where and how new particles are released into the environment and they can be analyzed both from their spatial and temporal behaviors. Besides well-defined geometric shapes such as point, line or circle sources, the geometry of the particle cloud to be created can be made dependent on the environment properties such as particle or field values. Also their temporal behavior can depend on the environment making it possible to

```
- apply: (id *) dom
{
    Particle *par;
    Field *field;
    point3d *pos, *vel;

    par = dom[0];
    field = dom[1];
    pos = [par position];
    [field valueFor: pos into: vel];
    [par setVelocity: vel];
}
```

Figure 1: Example of an interaction apply method

create other sources besides continuous or periodic ones.

The toolkit sends rendering messages to the objects that require a visual representation. Rendering is performed issuing Renderman commands [8] and the user can select the level of detail.

3. System Architecture

The SiPaViS toolkit is divided in two main layers as shown in Fig. 2. The application layer is responsible for dealing with user generated events. These events are then translated into messages sent to the toolkit kernel. To incorporate the toolkit in a new application, a different *Event Manager* object is all that has to be developed. The SiPaViS toolkit kernel is, in turn, composed by a collection of objects with distinct functionality. The *Controller* object receives events from the application and, in response, controls the behavior of the other toolkit objects. The *Parser* loads and interprets the simulation scripts while the *Class Manager* deals with loading and unloading dynamic classes.

The *Particle*, *Interaction*, *Field* and *Source* blocks represent the four abstract classes corresponding to the four different types of user defined objects. The kernel has no knowledge of user-defined classes, as they are dynamically loaded, and communicates with them through the interface defined by the abstract classes.

In common with dataflow visualization environments is the extensibility and modularity of SiPaViS by adding new classes of objects and/or connecting them in different ways. The memory requirements are kept to a minimum since only the classes needed in a particular problem are loaded during the script processing stage.

However, the granularity is much finer, also contributing to much lighter memory requirements. Particles need only to access data in a small neighborhood around their positions, as opposed to, at least, one module having to process all the data in dataflow environments.

The ODE solver developed for SiPaViS is used to integrate particles' trajectories, offering numeric integration techniques from the simple EULER's method to the more powerful RUNGE-KUTTA's methods. Essential to this task is the work done by interactions applying forces to particles. However, our ODE solver can also be used to integrate any other particles' attributes as long as an interaction is available to compute their time derivatives. This generality is achieved by sending messages to particles to collect their current state and derivative vectors.

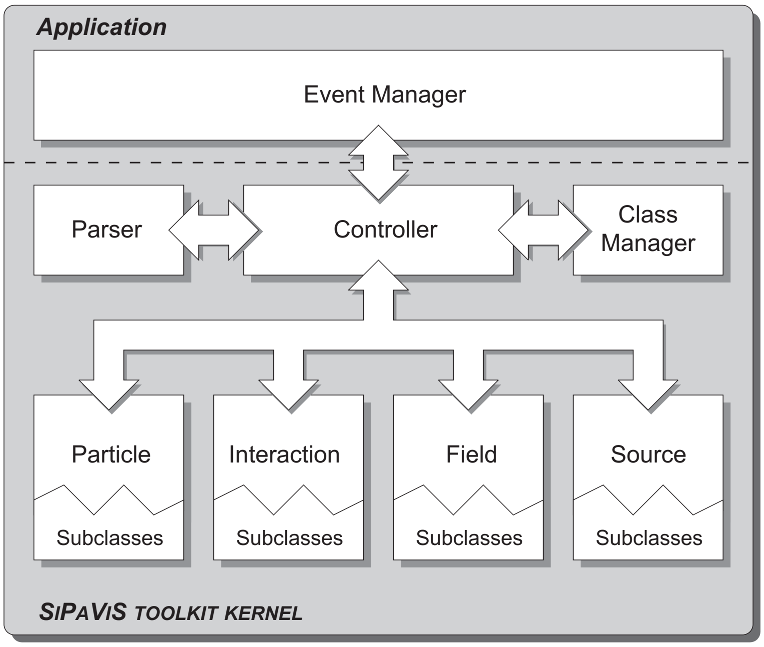

Figure 2: SiPaViS toolkit architecture

Every object in the toolkit can receive temporal notification messages as a means to synchronize them with the simulation clock. Only instances from those classes implementing the correspondent method will receive these notifications. Particles can use this mechanism to limit their lifetime or to maintain history records such as past trajectories. Time dependent fields can also be easily implemented by capturing these messages. On the other hand, particle sources are special cases because they heavily rely on this mechanism to know when to inject new particles, so it is not possible for them to ignore temporal notifications.

SiPaViS also allows for an object to be notified when a particle is about to die. Perhaps the most common use for this is to create particle replacements by particle sources. Usually, a particle dies when it is no longer needed. However, some of their attributes may need to be collected by other objects before removing the particle from the system.

4. Applications

In this section we will show how the toolkit can be used to develop several visualization algorithms applied both to scalar and vector fields.

Image space volume rendering is usually achieved by the ray-casting technique. This algorithm can be implemented in our toolkit by placing a particle source that releases a rectangular grid arrangement of particles through a scalar field. At least one particle should be assigned to each grid point (pixel) and each particle will behave as a ray with color, opacity and pixel coordinate attributes.

Instead of having the particle source releasing all particles in the beginning of the simulation, we can take benefits from death notifications. This way, the particle source will release a small square of particles each time. And only when all particles die will new ones, in the next square, be created. This improves the locality of accesses to the data set making it possible to exploit the cached virtual memory service provided by the toolkit. This service allows for a client to map a very big amount of memory and control the portion (cache) needed to be

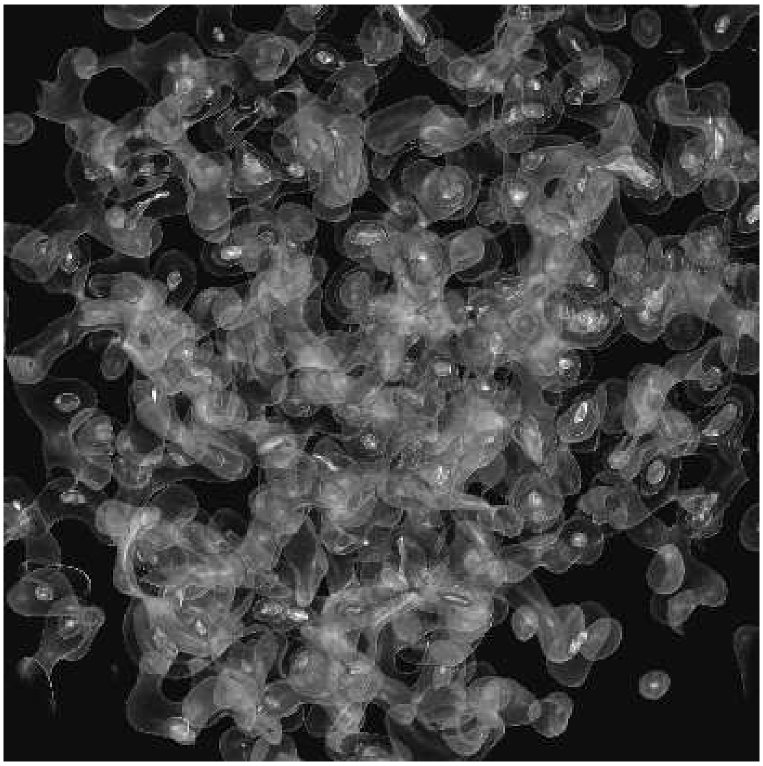

Figure 3: Different isosurfaces from an electron density map data set

loaded into main memory. The field class used to produce Figures 3 and 4 uses this service and the data set is subdivided into sub-volumes, each with $16 \times 16 \times 16$ contiguous samples.

In ray-casting there are no interactions applying forces to particles since they will move through the volume at constant speed and direction. In this case, interactions are only required to retrieve the field values at the particles' positions and compute the transfer functions needed to accumulate color and opacity values. When a particle opacity value reaches a maximum defined value the particle is killed, since its major contribution to the final pixel value is already computed [3].

It is important to notice that interactions are also valuable for speeding up the algorithm. Initially, all particles are either accelerated to their first intersection with the volume boundaries, or immediately killed if no intersection is found. This is accomplished by providing an interaction with its domain consisting of the particle class used for the rays and the field object to be visualized. Each time this interaction is applied it will test for intersections between one particle trajectory and the volume boundaries and act as explained before.

In Fig. 3 different isosurfaces are visible, showing clumps of atomic density and individual atoms. The data set consists of an electron density map of active site of superoxide dismutase (SOD) enzyme as determined by X-ray crystallography.

Fig. 4 is a volume rendering of the air-skin interface in a CT scan of a human cadaver head. The gradient of the scalar field was used to estimate the surface normal vectors required to compute direct illumination. It should be noted that data corruption visible in the image results from scattering of X-rays due to dental fillings. Particles used to compute the left side of the image are identical to the ones used on the right side. However different interactions were applied to display different features of the data set. On the right side the outermost surface was made partially transparent to let the inner surface become visible.

Figure 4: Air-skin interface rendered from a CT scan (256 × 256 × 113 samples) of a human cadaver head

Traditional particle systems simulations are also possible within our framework. Fig. 5 shows an example of behavioral animation[7], where a top view of a pool populated with fishes can be seen. Fishes are modeled as particles and they avoid collisions with each other and with environmental obstacles, while trying to swim in schools.

Vector field visualization techniques commonly use particles flowing through the volume. In CFD applications the volume is often a velocity field where particles are released. Fig. 6 shows several vector field visualization techniques simultaneously. An interaction is used to compute each particle's velocity vector from an helicoidal vector field and programmed as shown in Fig. 1. The complete declaration of the particle sources involved is given below.

```
Tube: GCircle(howMany n speed) {}
      {transformations_list}
      Flash(t0) TracePath(maxSteps)
Ribbon: Gline(howMany n speed) {}
        {transformations_list}
        Flash(t0) TracePath(maxSteps)
Animation: Spot(x y z) {}
           {transformations_list}
           Continuos(t0 freq)
           GlowSphere(radius)
```

The light gray (yellow in the original color picture) tubes in Fig. 6 are stream tubes obtained from a circular particle source belonging to class `GCircle`. The `howMany` parameter for this type of object specifies the number of particles released at each activation. The source keeps a pointer to each particle it creates and it can remember the last `n` activations. Instead

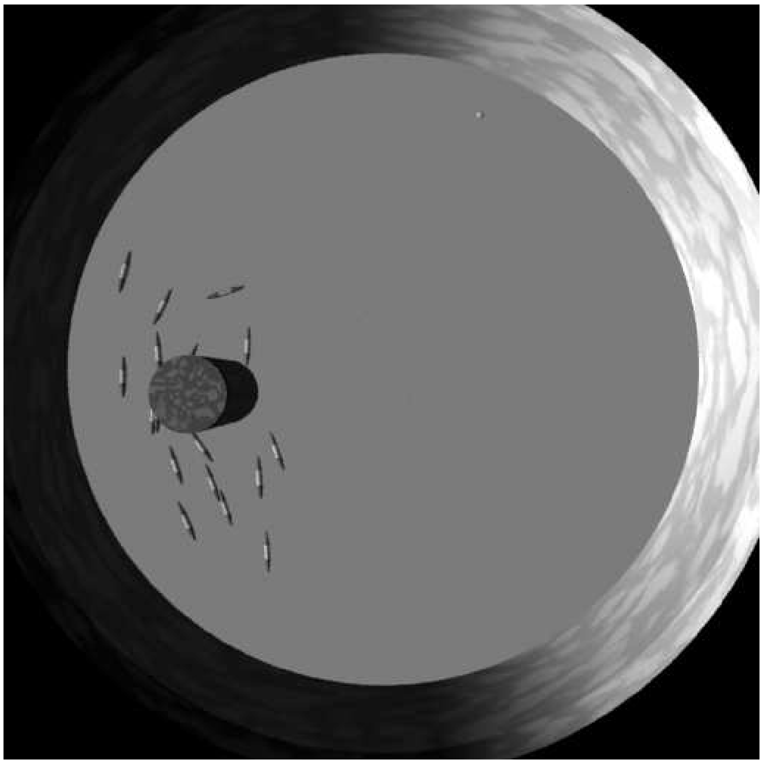

Figure 5: School of fishes in a pool

of rendering individual particles we render the particle source by displaying the polygonal mesh connecting all particles at their successive locations. In this case particles are created only once. If a periodic temporal source component was used, we would obtain a succession of stream tubes because the trajectory memorized by each particle is limited to a few (`maxSteps`) time steps.

In Fig. 6, the stream ribbon, located between the stream tubes, is formed using a similar technique but, instead of a circular source, we used a particle source releasing particles in a line segment (`Gline`). Each particle is assigned a different color to enhance perception of rotational movements. Nevertheless, its color could be used to code other field properties. The individual particles (blue in the original) show simple particle animation and they were created using a continuous temporal source with frequency given by the `freq` parameter.

It is highly important to notice that the type of particles created by each source is also a parameter in the source declaration. This way the same particle source can be used to create different types of particles. Not less important is the fact that particle source objects are created by grouping two different components: one providing its temporal behavior and the other controlling the geometry and location of the particle cloud. The classes `Flash` and `Continuous` are examples of temporal components while classes `GCircle`, `GLine` and `Spot` are spatial components.

The empty braces at the end of the first line in each source declaration can be used to specify a domain for each source object. This domain is a list of objects that may be inquired by the particle source to determine its characteristics. It is thus possible, for instance, to create sources that generate particles in a neighborhood of an already existent particle. Another possibility is a source whose particle cloud's density is controlled by a field value at a specific location.

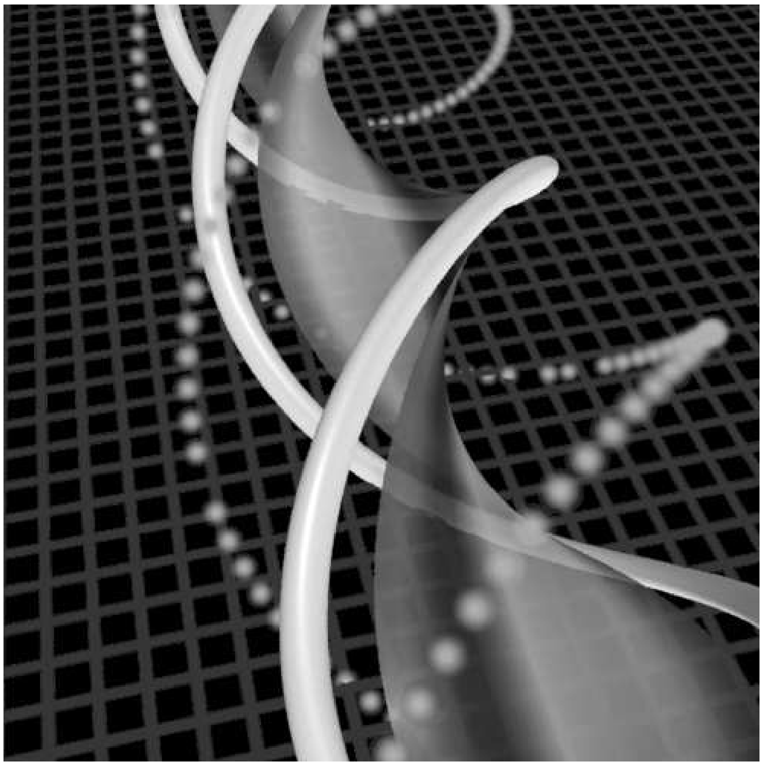

Figure 6: Mixing different vector field visualization techniques

5. Conclusions

The main advantages of using SiPaViS are the quick development of visualization and simulation algorithms over a broad area of applications. We have showed some examples from scalar and vector field visualization techniques as well as traditional particle systems simulations such as behavioral animation. Algorithms for isosurface extraction, where particles interact not only with their surrounding fields but also with other particles [2], and techniques to model fuzzy objects such as fire, smoke or explosions [6], are also easily implemented in our framework.

The object-oriented approach effectively accelerates the development of new techniques since the nuclear functionality is already coded inside the abstract classes. As the collection of programmer defined classes tends to increase, further development of new techniques becomes easier. Connecting objects in a different manner or making minor modifications to already existent classes may produce radically new effects.

The generality of the ODE solver implemented in the toolkit also allows the programmer to extend the base particle model in a more accurate manner since all new attribute values may be integrated over time. As an example it is thus possible to create particles with a complete coordinate system including orientation.

We think this approach is a good alternative to several visualization techniques no easily implemented in netwok-based tools. Memory limitations, commonly found in dataflow visualization environments, are also easier to overcome. This is mainly due to the finer granularity of the algorithms, as a consequence of the use of particles as visualization agents. Also, the dynamic class loading mechanism offered by SiPaViS alleviates memory requirements.

Acknowledgments

Data sets used to produce Figures 3 and 4 were obtained from the University of North Carolina — Chapel Hill volume rendering test data sets. All the work is being mainly supported by the FCT Foundation under the project PRAXIS/P/EEI/10089/1998.

References

[1] F. Birra: *Sistema de partículas para visualização e simulação (Particle system for visualization and simulation).* M.Sc. Dissertation (in Portuguese), New University of Lisbon, 1997.

[2] P. Crossno, E. Angel: *Isosurface extraction using particle systems.* IEEE Proceedings of Visualization '97, 495–498 (1997).

[3] M. Levoy: *Efficient ray tracing of volume data.* ACM Transactions on Graphics **9**, no. 3, 245–261 (1990).

[4] A. Pang: *Spray Rendering.* IEEE Computer Graphics and Applications **14**, no. 5, 57–63 (1994).

[5] L. Pinson, R. Wiener: *Objective C: Object Oriented Programming techniques.* Addison-Wesley Publishing Co., Reading, Mass., 1991.

[6] W. Reeves: *Particle systems — a technique for modeling a class of fuzzy objects.* Computer Graphics (Proceedings of Siggraph) **17**, no. 3, 359–376 (1983).

[7] C. Reynolds: *Flocks, herds and schools: A distributed behavioural model.* Computer Graphics **21**, no. 4, 25–34 (1987).

[8] S. Upstill: *The Renderman Companion.* Addison-Wesley Publishing Co., Reading, Mass., 1990.

Received August 14, 1998