1995

Distributed and Collaborative Synthetic Environments

Chandrajit L. Bajaj

Fausto Bernardini

Report Number:

95-059

DISTRIBUTED AND COLLABORATIVE SYNTHETIC ENVIRONMENTS

Chandrajit L. Bajaj
Fausto Bernardini

Department of Computer Science
Purdue University
West Lafayette, IN 47907

CSD-TR-95-059
September 1995

DISTRIBUTED AND COLLABORATIVE SYNTHETIC ENVIRONMENTS

Chandrajit L. Bajaj and Fausto Bernardini
Department of Computer Science
Purdue University
West Lafayette, IN 47907
{bajaj,fxb}@cs.purdue.edu

Introduction and Overview

Fast graphics workstations and increased computing power, together with improved interface technologies, have created new and diverse possibilities for developing and interacting with synthetic environments[1,2,3,4,5,6]. A synthetic environment system is generally characterized by the following components:

- Input/output devices, that constitute the interface between the human senses and the synthetic environment generated by the computer. Several degrees of immersion are possible, ranging from simple stereoscopic view of an image on a CRT display to a total immersion in which a head-mounted display, sound and haptic devices (force and torque feedback, tactile stimuli) are used.
- A computation system, running a real-time simulation of the environment. This can sometimes be subdivided in several subsystems, for example a simulation module running on a supercomputer coupled with a scene creation and a rendering module running on a graphics workstation.

To achieve an acceptable level of realism, the display subsystem must generate at least 10 frames per second of high quality graphics. For complex environments, this goal is still far from being reached. Nonetheless, synthetic environments have already been applied successfully in such diverse fields as: Operations in hazardous or remote environments, through telepresence; molecular modeling; flight simulations; battlefield simulation; architectural walk-throughs; surgical planning; collaborative design; education and training; entertainment, and many others.

A basic need of a synthetic environment system is that of giving to the user a plausible reproduction of the visual aspect of the objects he is interacting with. It is well known that populating a synthetic world with objects, be these spaceships in a science fiction movie, prototypes in a manufacturing design or replicas of existing real-world objects in an architectural walk-through, can be extremely time consuming. Not only it must be possible for the users to move in the environment, updating in real time his or her view of it. To convey the impression of an immersive, active presence in an environment, a real-time simulation of the physical behavior of the various components of the environment is necessary. Moreover, in many applications the users should be allowed to interrogate objects about their associated properties, and to interact with them, for example modifying some of their properties to support *what-if* scenarios.

The goal of our Shastra research project is that of providing a substrate of geometric data structures and algorithms which allow the distributed construction and modification of the environment, efficient querying of objects attributes, collaborative interaction with the environment, fast computation of collision detection and visibility information for efficient dynamic simulation and real-time scene display. In particular, we address the following issues:

- A geometric framework for modeling and visualizing synthetic environments and interacting with them. We highlight the functions required for the *geometric engine* of a synthetic environment system.
- A *distribution* and *collaboration* substrate that supports construction, modification and interaction with synthetic environments on networked desktop machines.

Geometric Engine

A critical subsystem in all synthetic environment systems is the *geometric engine*, or the software module responsible for creating a realistic view of the simulated world, and for allowing the user to interact with it. In a typical scenario, a system requires the display of several objects, that move and must behave realistically. A user wanders in the environment, constantly changing his or her point of view. Other users (or actors) may be sharing the same environment, and a suitable representation of them could be required. The users interact with objects in the environment, for example touching their surface to pick and query an object, grabbing and moving them, or simply colliding with a wall of a room. Simulation of even the simplest form of dynamic and interaction requires collision detection and contact analysis. Fast display of complex environments requires efficient visibility computation. Querying and interacting with objects requires rapid point location and local shape control. In short, the geometric engine of a general-purpose synthetic environment system should provide efficient data structures and algorithms for:

- Shape representation
- Dynamic object insertion and deletion
- Object animation (motion, non-rigid transformation)
- Object location and closest point queries
- Collision detection and contact analysis
- Visibility ordering and culling
- View-volume clipping
- Multi-resolution representation
- Real-time high-quality rendering
- Other operations required by specific applications (e.g., set operation and interactive shape control for collaborative design)

We want to explore the use of data structures suitable to support the operations listed above in a complex, constantly changing environment. Several types of data structures that partially satisfy these needs have been proposed in the past. However, they have mainly been used in Computer Aided Design (CAD) systems or other special-purpose applications, and not as an infrastructure for a complex, dynamic and general-purpose environment. Often, CAD systems use boundary representations (Breps) to describe the geometry of the object being modeled. Breps are well suited to the implementation of the most common modeling operations. However, when this representation is to be used in a general-purpose synthetic environment, it has to be supplemented with additional structures such as octrees or bounding volumes hierarchies to achieve the required efficiency.

Several variants of the *octree* data structure have been proposed, both as a superimposed search index on existing representations of geometry, or as the main representation scheme in the system[7]. The simplest variant (and usually the one requiring the most space) is the *region octree*. In a region octree, a cubic domain is split at each node in eight equal sub-cubes. The decomposition continues recursively for each node that requires a further spatial refinement. Leaves of the octree represent empty or solid regions of space. The major drawbacks in the use of a region octree for representing the geometry are the fact that it is an approximation (unless the object is constituted by mutually orthogonal planar faces only), and its size. Another octree variant is the *PM octree* (PM stands for Polygonal Map), each leaf corresponds to a single vertex, edge or face. The only exception is that a leaf may contain more than one edge (face) if all edges (faces) are incident on the same vertex. A PM octree usually requires much less storage than a region octree[8,9], and it permits an exact representation of polyhedral models. PM octrees support a large suite of modeling operations, and several methods are known to convert CSG or Breps models to PM octrees and vice-versa[7].

A *K-d-tree*[10] (where *K* is the dimension of the domain space, in our case we will deal with 3-d-trees) is a binary tree in which internal nodes partition the space by a cut hyperplane defined by a value in one of the *K* dimensions, and external nodes, or *buckets*, store the points in the resulting hyper-rectangles of the partition. They allow *O(log n)* insertion, deletion, and point query operation. The semidynamic variant introduced in[11] allows constant expected time deletion, undeletion, nearest neighbor searching and fixed-radius near-neighbor searching. When used to model the geometry of an object, they suffer of some of the same disadvantages of region octrees in that they can only approximate the geometry of general objects. However, they have been successfully used in precomputing viewpoint-independent visibility and illumination information for large models of buildings, to speed up a successive interactive walk-through phase[12,13,14]. Changes in the environment would require a new processing of the visibility information.

Binary Space Partitioning Trees (BSPT)[15] are a generalization of *K*-d-trees. In a BSPT, the cutting planes associated to the internal nodes are not constrained to be orthogonal. Constructing a BSPT representation of one or more polyhedral objects involves encoding the polygonal faces into a binary tree of cutting planes. Notice that affine transformations on the encoded object or groups of objects do not change the tree structure, but requires only the application of a transformation to the plane equations. The spatial ordering encoded in the tree can be exploited for the speedup of intersection and visibility computation. Moreover, when a BSPT is properly constructed, it offers a *multi-resolution* representation of the object: As one descends a path in the tree, the bounded region decreases its size monotonically. The effectiveness of this technique in practice has been proved in several applications.

Unstructured tetrahedral grids are extensively used in finite element analysis, and triangulations are pervasive in computational geometry. The use of simplicial complexes as a general approach to representing geometric shape has been advocated by several authors[16,17]. We are currently investigating the use of *3D Regular* (a *weighted* variant of Delaunay) *triangulations*, coupled with efficient point location data structures, to provide the functionalities needed in a synthetic environment. A representation based on Hierarchical Simplicial Complexes has been recently proposed[18,19]. (see also an extension to a hierarchy of cell complexes[20]). In our scheme, the top-level triangulation subdivides the space in tetrahedral cells (with the associated search structure). Each cell contains an object (or a group of objects), and these are described by other triangulations. The scheme can be recursively used to achieve the level of detail in the decomposition needed by the application. Using a hierarchy of triangulations has several advantages. Regular triangulations are acyclic with respect to visibility ordering[21]. This means that, given a viewpoint, it is possible to order the cells of the complex in a sequence such that if A obstructs B, then B precedes A in the sequence. This property is preserved in a hierarchy defined as above. Moreover, a hierarchy of simplicial complexes naturally defines a multiresolution representation of the geometry, and allows fast point location and other types of queries when associated with appropriate data structures.

A 3D Regular triangulation can be built incrementally, via point insertion and topological flipping. A history DAG can be used in the incremental algorithm to allow the efficient location of which tetrahedron in the current triangulation the point to be inserted lies in, as well as to help in other point location queries required by the application. Moreover, a *Power* (weighted Voronoi) *diagram* (eventually of order k), can be easily computed to allow fast answers to closest point queries. We are considering the use of several variations of this approach. A possible version of the data structure consists in using a Delaunay triangulation that conforms to objects boundaries. When a new object is inserted in the environment, the triangulation is updated to accommodate all its faces. Methods to construct conforming Delaunay triangulations in 2D and 3D by inserting extra points are known[22,23,24,25,26]. However, while a polynomial bound on the number of extra points is known for the two-dimensional case ($O(m^2n)$, for m edges and n vertices[27]), whether such a bound exists for the 3D case is still an open problem. Paper[28] reports statistics from experiments with the 3D algorithm that show a reasonable behavior on models used in real applications. This scheme is restricted to polyhedral objects (curved surface objects can be approximated by small planar faces).

In a different scheme, the triangulation is used as the domain for implicit piecewise-algebraic surfaces[29,30,31,32,33]. In each tetrahedron containing a piece of some object's boundary, a polynomial function $f(x,y,z)$ of degree n (where n is usually small, often $n \leq 3$) is defined, and the piece of surface is implicitly given by $f(x,y,z) = 0$. The surface patches can be made to join with some degree of continuity (for example C^1 or C^2), by using interpolants of appropriate degree. For $n = 1$, in particular, the surface pieces are planar polygons, and the representation resembles the PM octree, where the cubic cells are replaced by tetrahedral ones. This approach has the advantage of allowing a compact representation for curved objects. Moreover, it makes possible to use a more "relaxed" condition on the conforming triangulation. This needs not to conform to the object boundary, but simply contain a set of tetrahedra suitable to define the piecewise surface. An implicit piecewise-algebraic surface is suitable for interactive local control. In a 3D synthetic environment, one could very naturally make use of 3D widgets to allow an intuitive control on the surface shape. For example handles could be attached to the surface that can be pulled or pushed to deform it.

Networked Distribution and Collaboration

We advocate the approach of integrating a collection of function-specific tools into a distributed and extensible environment, where tools can easily use facilities provided by other tools[34],[35],[36].

Isolation of functionality makes the environment modular, and makes tools easy to develop and maintain. Distribution lets us benefit from the cumulative computation power of workstation clusters. Tool-level cooperation allows us to exploit the commonality that is inherent to many scientific manipulation systems. An enabling infrastructure of communication and interaction tools, display and visualization facilities, symbolic processing substrates, and simulation and animation tools saves avoidable re-implementation of existing functionality, and speeds up the application development.

The Shastra environment consists of multiple interacting tools. Some tools implement scientific design and manipulation functionality (the Shastra Toolkits). Other tools are responsible for managing the collaborative environment (Kernels and Session Managers). Yet others offer specific services for communication and animation (Service Applications). Tools register with the environment at startup, providing information about the kind of services that they offer (Directory), and how and where they can be contacted for those services (Location). The environment supports mechanisms to create remote instances of applications and to connect to them in client-server or peer-peer mode (Distribution). In addition, it provides facilities for different types of multi-user interaction ranging from master-slave blackboarding (Turn Taking) to synchronous multiple-user interaction (Collaboration). It implements functionality for starting and terminating collaborative sessions, and for joining or leaving them. It also supports dynamic messaging between different tools. Tools are thus built on top of the abstract Shastra layer, which is depicted in Figure 1.

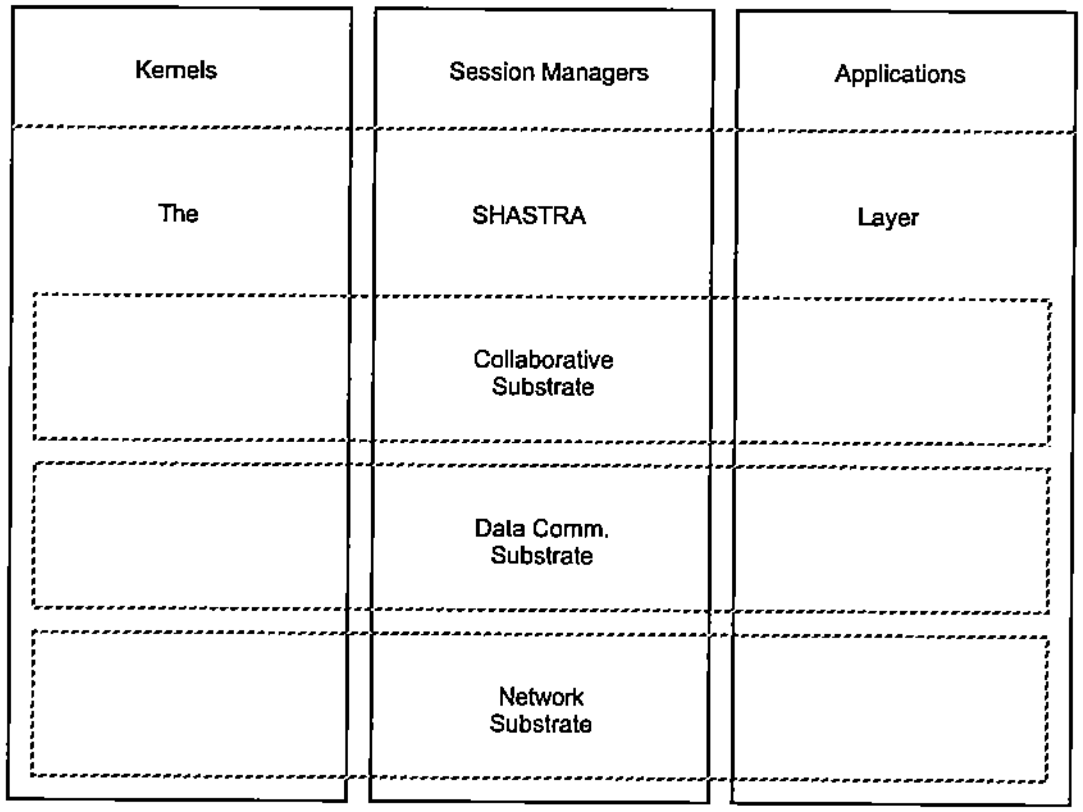

Figure 1 The Shastra Layer: A connection, communication and collaboration management substrate. Shastra tools inter-operate using facilities provided by this layer.

The SHASTRA collaborative scientific environment provides mechanisms to support a variety of multi-user interactions spanning the range from demonstrations and walk-throughs, to synchronous multi-user collaboration. In addition, it facilitates synchronous and asynchronous exchange of multimedia information which is useful to successfully communicate at the time of design, and to share the results of scientific tasks, and often necessary to actually solve problems. The infrastructure provides facilities to distribute the input of low computation tasks - to obtain the parallelism benefit of distribution, and the output of compute intensive tasks - to emphasize sharing of resources among applications. It provides a convenient abstraction to the application developer, shielding him from lower level details, while providing him with a rich substrate of high level mechanisms to tackle progressively larger problems.

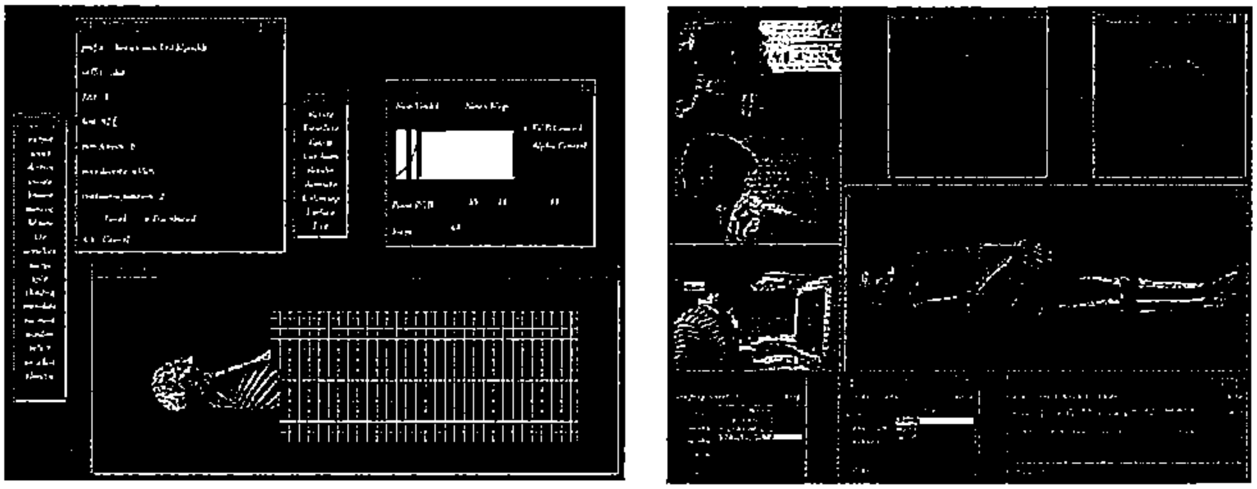

Figure 2 (left) *Distributed Volume Visualization from a large data set.* ***(right)*** *Shared Visualization. A group of researchers uses Sha-Poly to share volume visualization images of a head with cutaways (top center and right) and a cadaver (center).*

A teleconferencing approach to modeling and analysis of empirical data is presented in[37], where the authors hypothesize about a collaborative scientific visualization environment. A discussion of an interactive visualization environment for 3D imaging is presented in[38], where the authors adapt the electron microscope to perform as a computer peripheral. An environment like Shastra makes it convenient to build collaborative visualization and manipulation facilities, that support resource sharing in a distributed setting.

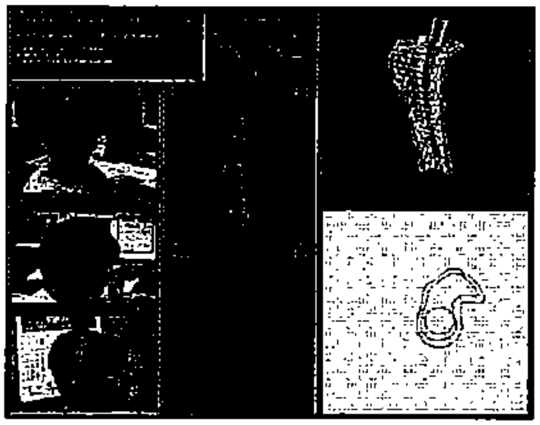
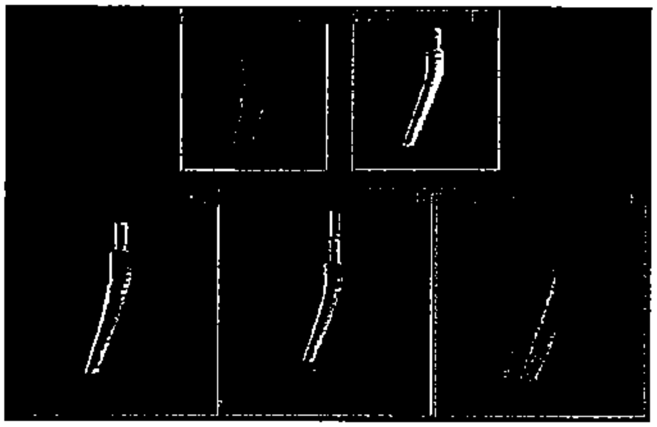

Figure 3 (left) Collaborative customized hip implant design. A designer uses the Shastra collaborative tool Shilp to interactively create a geometric model of a hip implant (top right) by generating cross-sectional contours of the implant (bottom center and right) from a sectional model of the femur (center) created in Vaidak (another toolkit of the Shatra suite). A video conference supports communication. (right) Several steps in the design optimization.

Shastra Architecture

Tools in Shastra are built with the underlying idea of inter-tool cooperation. Every tool is abstractly composed of three layers. The Core is accessed through any of the Interfaces *via* a Mapper. The application-specific Core implements the functionality offered by the tool. Above the Core is a functional Interface Mapper that invokes functionality embedded in the Core in response to requests from the Graphical User Interface, ASCII Interface or the Network Interface. It also maps requests to alter the user interface or to send messages on the Network Interface. The Mapper is essentially a command interpreter that invokes registered event handlers when events of interest occur. Tools register event handlers with the Mapper for events they are interested in, and unregister those that cease to be of interest. The separation of Core and Interface, that of function and interface, makes it easy to build multi-user systems, since it enables the maintenance and display of shared state at a user interface *via* remote commands in a distributed system.

The GUI is application-specific. The ASCII interface is a shell-like front end for the tool. Tools communicate with other tools in the environment, *via* the Shastra substrate, through an abstract Network Interface. This implements the underlying messaging system that provides connection and transport facilities. The Network Interface multiplexes multiple simultaneous network connections, and implements the different application level communication protocols. Functionality available at a network interface is described to the communication substrate using a signature that specifies callback functions for the different kinds of network events that the tool is interested in. The signature provides an abstract interface to remote systems, and describes functionality offered by the tool. It also serves as a regulatory mechanism, since different levels of service can be offered at different interfaces by specifying the appropriate signatures.

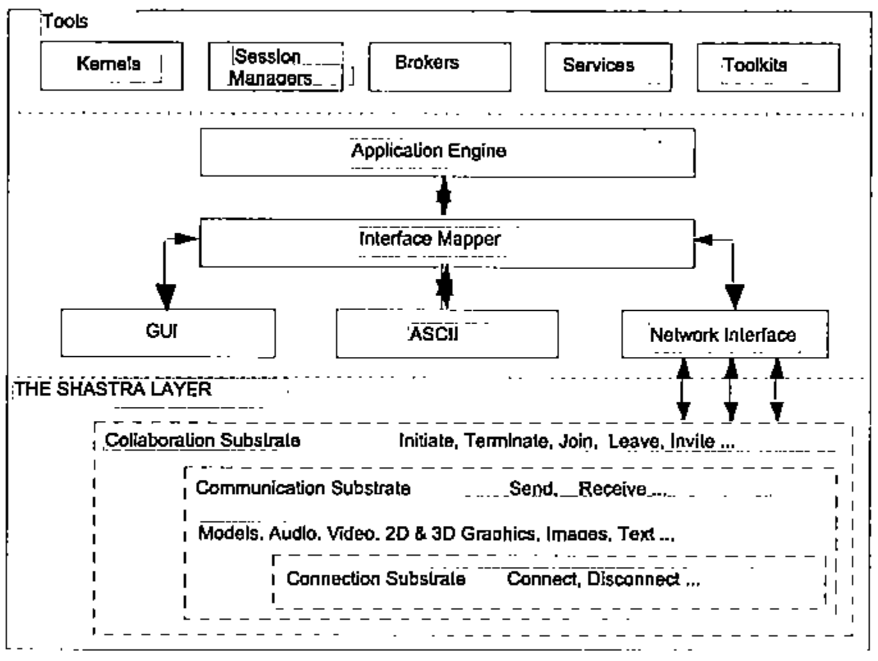

Figure 4 High Level Architecture of a Tool in the Shastra Environment.

To take advantage of the integration facilities of the infrastructure, the Core uses the Network Interface to access functionality already implemented in other tools. The main benefit from this setup is modularity and reuse - tools isolate the functionality they offer, and provide a functional interface to peers. The high level block architecture of tools in Shastra is depicted in Figure 4. The architecture makes it easy for tools to connect to other tools and request operations, synchronously as well as asynchronously.

These architectural guidelines accord us the benefit of uniformity since all tools are built upon a common infrastructure and have identical connection, communication and collaboration mechanisms. The concept of cooperation awareness thus pervades the architecture. The entire set of connected Network Interfaces of Shastra tools manifests itself as the abstract Shastra layer at runtime (see Figure 4). It maintains the collaborative environment, provides access to functionality of different systems, and provides facilities for initiating, terminating, joining, leaving and conducting collaborations. The connected network interfaces of Shastra tools comprise a distributed virtual machine on which we build problem solving applications.

The enabling substrates use the event paradigm to provide functionality. Tools use the application programming interface of the substrate to cause request messages to be sent over connections. Tools interested in any event register handler functions for it with the Mapper. The handler functions are invoked when that event is received. This allows tools to take action appropriate to the event when it occurs.

Session Model

In this model, a Session is the unit of collaborative activity. A Session is essentially a Context without an Interface. Session Model based collaborative tools are implemented in our Collaboration

Model by instantiating a Session, that causes the setting up of connected shared Contexts in multiple tools. These shared Contexts are collaborative task aware. Events that are associated with low computation tasks are routed to the Session Context, which relays them to all shared Contexts. Events that are associated with compute-intensive tasks are acted upon in the tool Context, and the associated Triggers are routed to the Session Context. Context State changes generate Triggers that are routed to tools and update views at their Interfaces. The implementation is depicted in Figure 5.

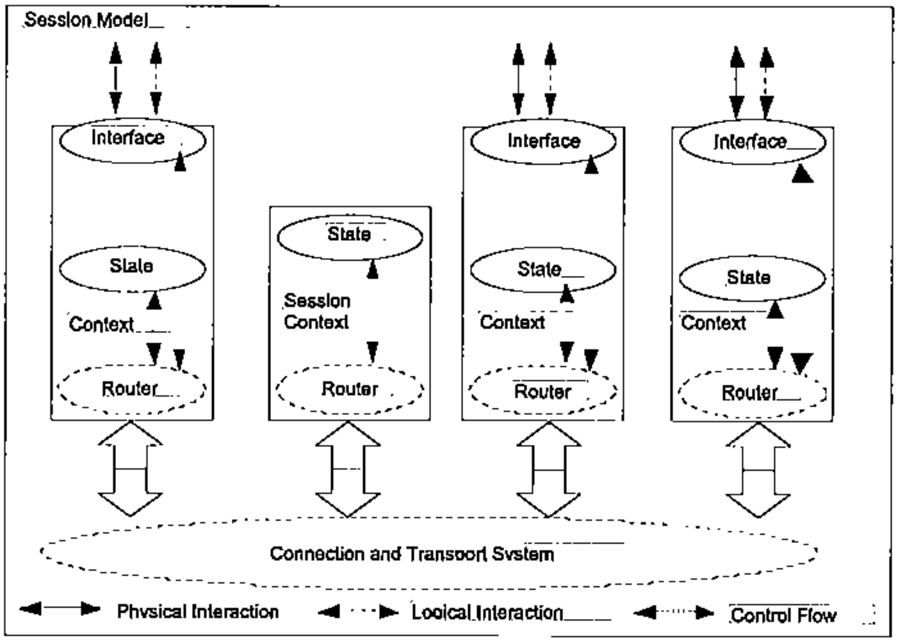

Figure 5 Session Model of Collaboration.

Since Sessions are collaborative task-aware, they can choose between centralized and replicated data management facilities based on the number of sites in the collaboration, degree of dependence between collaborative tasks, and performance of the underlying mechanisms. Collaborative tasks are thus implemented in the shared Context and State of a Session. Simultaneous interaction is supported from multiple distributed interfaces.

A major advantage of this approach is that Sessions can be made persistent, since they are delinked from user level tools and interfaces. They can be saved and restarted, and thus support asynchronous and synchronous collaborative interaction. Also, participating in collaborative tasks is further simplified, since tools do not have to keep track of group membership, or set up routing information. Tools create Contexts that are shared with the Session Context when they join a Session, and tear them down when they leave.

Runtime Environment

Collaborative Sessions, or Sessions, are instances of synchronous multi-user collaborations or conferences in the Shastra environment. A collaboration in Shastra consists of a group of cooperating tools regulated by a Session Manager, the conference management tool of Shastra. One Session Manager runs per collaborative session. It maintains the session and handles details of connection and session management, interaction control and access regulation. It keeps track of membership of the collaborative group, and serves as a repository of the shared objects in the collaboration. It supports a multicast facility needed for information exchange in a synchronous multi-user conferencing scenario. It has a constraint management subsystem that resolves conflicts that arise as a result of multi-user interaction, enabling maintenance of mutual consistency of operations. It has a regulatory subsystem that controls synchronous multi-party interaction, and provides a floor control facility based on turn-taking. Every Session Manager implements functionality to service the following session control requests.

- Invite - Request to invite a tool to an ongoing session.
- Join - Request to join an ongoing session.
- Remove - Request to remove a tool from a session.
- Leave - Request to leave a session that the tool is a member of.
- End - Request to terminate a collaborative session.

It also serves the following interaction control requests.

- Format - Request to set session format.
- Capabilities - Request to set access regulation capabilities.
- Interaction Mode - Request to set interaction mode for the session.
- Request Floor - Request to get floor control for the session.
- Release Floor - Request to release floor control for the session.
- Assign Floor - Request to assign floor control for the session.

A collaborative session in Shastra is started by a tool when it sends the Session Start message to the local Kernel. This causes the instantiation of a Session Manager for the incipient session. The initiating tool becomes the Session Leader. A tool sets session format using the Format message. Sessions may be Formal, where participation is by invitation only, or Informal where any tool can dynamically join the conference. The Leader assigns capabilities of other participants for collaborative activity in the session using the Capabilities message.

The interaction mode for a session is specified using the Interaction Mode message. Interaction can occur in the Regulated or Free mode. In the Regulated mode, tools request and relinquish the floor using Request Floor and Release Floor messages. The leader can explicitly assign floor control using the Assign Floor message. In the Free mode, interaction is regulated *via* capabilities assigned to session participants. Capabilities are described in a later section. Other tools are invited to participate in a session by sending them the Invite request *via* the Kernel. Tools can dynamically join ongoing sessions by sending the Join message to the relevant Session Manager *via* the Kernel. The Session Manager uses session format information to control dynamic incorporation of tools. The Leader can remove a participating tool from the session using the Remove message. Tools can discontinue participation in the session by sending the Leave message to the Session Manager. A session is terminated by the Leader using the End message.

Application-specific Session Managers for different collaborative tasks are created from the basic Session Manager that provides application independent connection, communication and collaboration control facilities. Such session managers support additional messages for collaborative operations specific to the application.

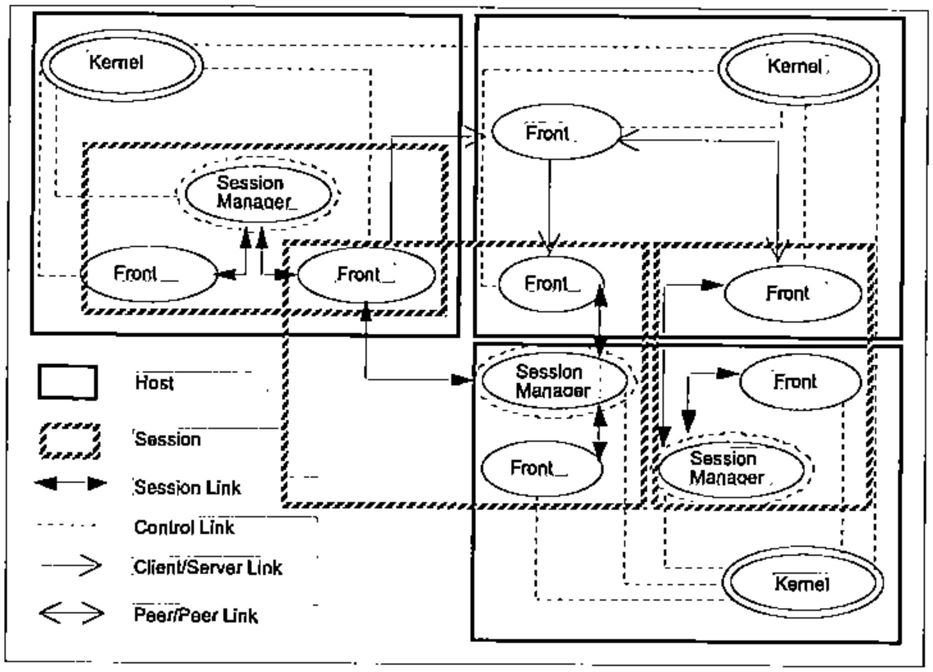

Figure 6 Information Flow in the Shastra Environment.

References

[1] MacDonald, L., and Vince, J. *Interacting with virtual environments.*. Wiley Professional Computing. Wiley, 1994.

[2] Thalmann, N. M., and Thalmann, D. *Virtual worlds and multimedia.* Wiley Professional Computing. Wiley, 1993.

[3] Wexelblat, A. *Virtual reality: applications and explorations.* Academic Press Professional, 1993.

[4] Special issue on Virtual Environments, IEEE Computer 28, 7 (July 1995).

[5] Special Isuue on Virtual Reality, IEEE Computer Graphics & Applications 14, 1 (Jan. 1994).

[6] Special issue on Virtual Reality, IEEE Spectrum 30, 10 (Oct. 1993).

[7] Samet, H. The *Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1990.

[8] Ayala, D., Brunet, P., Juan, R., and Navazo, I.,*Object representation by means of nominal division quadtrees.* ACM Trans. Graph. 4, 1 (1985), pp. 41-59.

[9] Brunet, P., and Navazo, I. *Solid representations using extended octrees.* ACM Trans. Graph. 9 (1990), 170-197.

[10] Bentley, J. L. *Multidimensional binary search trees used for associative searching*. Commun. ACM 18, 9 (1975), 509-517.

[11] Bentley, J. L. *K-d trees for semidynamic point sets*. Proc. 6th Annu. ACM Sympos. Comput. Geom. (1990), 187-197.

[12] Teller, S. J., and Sèquin, C. H. *Visibility preprocessing for interactive walkthroughs*. Comput. Graph. 25, 4 (July 1991), 61-69.

[13] Teller, S., and Hanrahan, P. *Global visibility algorithms for illumination computations*. In Proc. SIGGRAPH '93 (1993), pp. 239-246.

[14] Funkhouser, T. A., and Sèquin, C. H. *Adaptive display algorithm for interactive frame rates duringvisualization of complex virtual environments*. In Computer Graphics Proceedings (1993), Annual Conference Series. Proceedings of SIGGRAPH 93, ACM SIGGRAPH.

[15] Fuchs, H., Kedem, Z. M., and Naylor, B. *On visible surface generation by a priori tree structures*. Comput. Graph. 14, 3 (1980), 124-133.

[16] Paoluzzi, A., Bernardini, F., Cattani, C., and Ferrucci, V. *Dimension-independent modeling with simplicial complexes*. ACM Transactions on Graphics 12, 1 (Jan. 1993), 56-102.

[17] Edelsbrunner, H. *Modeling with simplicial complexes*. SIGGRAPH 94 Course Notes. ACM SIGGRAPH, July 1994.

[18] De Floriani, L., Falcidieno, B., and Pienovi, C. *Structured Graph Representation of a Hierarchical Triangulation*. Comput. Vision Graph. Image Process 45, 2 (Feb. 1989), pp. 215-226.

[19] De Floriani, L. *A Pyramidal Data Structure for Triangle-Based Surface Representation*. IEEE Computer Graphics & Applications 9 (Mar. 1989), pp. 67-78.

[20] Bertolotto, M., Bruzzone, E., and De Floriani, L. *Acyclic hierarchical simplicial complexes*. In Proc. 5th Canad. Conf. Comput. Geom. (Waterloo, Canada), pp. 279-284.

[21] Edelsbrunner, E. *An acyclicity theorem for cell complexes in d dimensions*. Proc. 5th Annu. ACM Sympos. Comput. Geom. 1989, pp. 145-151.

[22] Weatherill, N. P. *Integrity of geometrical boundaries in the two-dimensional Delaunay triangulation*. Communications. In Applied Numerical Methods 6, 2 (Feb. 1990).

[23] Edelsbrunner, H., and Tan, T. S.*An upper bound for conforming Delaunay triangulations*. In Proc. 8th Annu. ACM Sympos. Comput. Geom. (1992), 53-62.

[24] Hansen, A. J., and Levin, P. L. *On conforming Delaunay mesh generation*. Adv. Engineering Software 14, 2 (1992), pp. 129-135.

[25] Tan, T. S. *An optimal bound for quality conforming triangulations*. In Proc. 10th Annu. ACM Sympos. Comput. Geom. (1994),

[26] Weatherill, N. P. *Efficient three-dimensional delaunay triangulation with automatic point creation and imposed boundary constraints*. International Journal for Numerical Methods in Engineering 37,(June 1994), 2005-2039.

[27] Edelsbrunner, H., Tan, T. S. *An upper bound for conforming Delaunay triangulations*. Proc. 8th Annu. ACM Sympos. Comput. Geom. 1992, pp. 53-62.

[28] Weatherill, N. P., *Efficient three-dimensional Delaunay triangulation with automatic point creation and imposed boundary constraints*. International Journal for Numerical Methods in Engineering 37, 12 (June 1994), pp. 2005-2039.

[29] Bajaj, C., *Surface fitting with implicit algebraic surface patches*. Topics in Surface Modeling, H. Hagen, Ed. SIAM Publications, pp. 23-52.

[30] Bajaj, C. *The emergence of algebraic curves and surfaces in geometric design*. Directions in Geometric Computing, R. Martin, Ed, Information Geometers Press, 1993, pp. 1-29.

[31] Bajaj, C., Bernardini, F., and Xu, G. *Adaptive reconstruction of surfaces and scalar fields from dense scattered trivariate data*. Computer Science Technical Report CSD-TR-95-028, Purdue University, 1995.

[32] Bajaj, C., Bernardini, F., and Xu, G. *Automatic reconstruction of surfaces and scalar fields from 3D scans*. Computer Graphics Proceedings, (1995), Annual Conference Series. Proceedings of SIGGRAPH 95, ACM SIGGRAPH., pp. 109-118.

[33] Bajaj, C., Chen, J., and Xu, G. Modeling with cubic A-patches. ACM Transactions on Graphics 1995 (To appear).

[34] Anupam, V., and Bajaj, C. *Collaborative Multimedia in Scientific Design*, IEEE Multimendia Journal 1, 2 (Summer 1994), pp. 39-49.

[35] Anupam, V., and Bajaj, C. *Shastra: An architecture for development of collaborative applications*, International Journal of Intelligent and Cooperative Information Systems 3, 2 (1994), pp. 155-166.

[36] Anupam V., Bajaj, C., and Schikore, D. *Distributed and collaborative volume visualization*, IEEE Computer 27, 7 (1994), pp. 37-43.

[37] Carlbom, I., Hsu, W., Klinkner, G., Szeliski, R., Waters, K., Doyle, M., Gettys, J., Harris, K., Levergood, T., Palmer, R., Picart, M., Terzopoulos, D. Tonnesen, D., Vannier, M. and Wallace, G. *Modeling and analysis of empirical data in collaborative environments*, Comm. of the ACM 41, 6 (1992), pp. 73-84.

[38] Mercurio, P., Elvine, T., Young, S., Cohen, P., Fall, K., Ellisman, M. The distributed laboratory. Comm. of the ACM 41, 6, (1992), pp. 54-63.