Shastra: Multimedia Collaborative Design Environment

Vinod Anupam and Chandrajit L. Bajaj
Purdue University

A distributed environment that supports multimedia interfaces, Shastra strengthens collaboration in scientific and engineering design. Implemented on a desktop system, it provides an infrastructure for user- and application-level cooperation. We describe the Shastra architecture and tools, then walk through a scenario that shows how it enhances collaborative problem solving.

Today's multimedia desktop systems are powerful tools that can revolutionize the way we collaborate in scientific and engineering settings. Specifically, these systems let us depart from the traditional single-user design and analysis environment by supporting multiuser collaboration. The objective of our work is to develop next-generation design environments, where a geographically distributed team can create, share, manipulate, analyze, simulate, and visualize complex 3D geometric designs over a heterogeneous network of workstations and supercomputers.

Our approach integrates a collection of function-specific tools into a distributed, extensible environment. We provide an enabling infrastructure for collaboration. This infrastructure saves users from having to implement functionality already existing in the environment, thus speeding up application development. We also provide mechanisms to support a variety of multiuser interactions.

In this article, we first describe our approach and its implementation, then present an example design scenario using it.

Shastra design environment

Shastra (a Sanskrit word meaning "branch of knowledge") is a geometric and scientific design environment—that is, it provides facilities for geometric design and for simulation, visualization, and animation. At its core, Shastra features a *collaboration substrate* that supports synchronous multiuser applications, and connection and communication *distribution substrates* that emphasize distributed problem solving for concurrent engineering (see Figure 1 on the next page). These substrates are function libraries with well-defined abstract programming interfaces that establish a framework for session management, data sharing, and multimedia communication.

Fully integrated, Shastra supports rapid prototyping and development of software tools for the creation, manipulation, and visualization of multidimensional geometric data. The sidebar on page 41 compares Shastra to related work in engineering and scientific collaboration.

The Shastra runtime environment consists of multiple interacting tools, which are implemented on top of the collaboration, connection, and communication substrates as shown in Figure 1. Some tools manage the collaborative environment (kernels and session managers). Other tools implement scientific design and manipulation functionality (toolkits). Yet others offer specific services for communication and animation (services). These tools register with the environment at startup, providing information about the kind of services they offer (directory) and how and where to find them (location).

The Shastra environment provides the user facilities 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 multiuser interaction ranging from master-slave blackboarding (turn taking) to synchronous multiuser 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.

Architectural features

The Shastra design embodies the simple idea that geometric design and scientific manipulation toolkits can be considered as objects that provide specific functionality. These objects exchange messages—automatically or under user control—to request that other objects perform operations on their behalf. We present salient features of the Shastra architecture here (for a detailed description, see Anupam and Bajaj[1]).

At the system level, Shastra specifies architectural guidelines and provides communication facilities so that toolkits can cooperate and exchange information. At the application level, it provides collaboration and multimedia facilities so that users can develop applications cooperatively to solve problems. Shastra integrates these levels to let users design sophisticated problem-solving virtual machines.

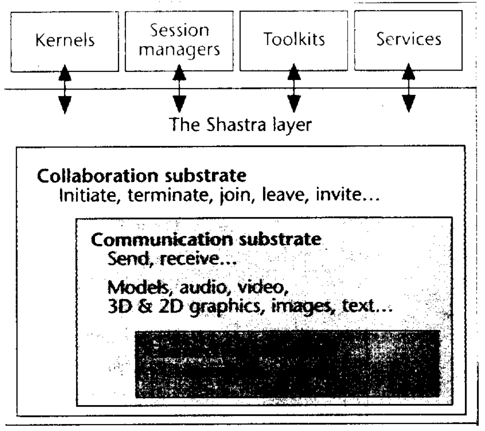


Figure 1. The Shastra layer. Shastra tools interoperate using facilities provided by the Shastra layer—a connection, communication, and collaboration management substrate.

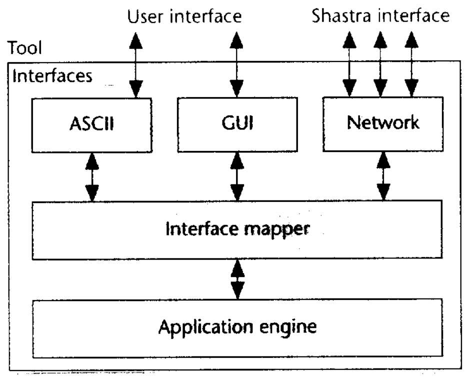


Figure 2. High-level block architecture of a Shastra tool. Users interact with tools via the user interface. Tools interact simultaneously with multiple tools via the network interface.

Figure 2 depicts the high-level block architecture common to all the tools in Shastra. This architecture makes it easy for tools to connect to each other and request operations, synchronously as well as asynchronously. A tool has an application-specific core, the application engine, which implements the tool's core functions. Above the core is a functional interface mapper, which invokes functions embedded in the engine in response to requests from the graphical user, ASCII, or network interfaces. The GUI is application-specific. The ASCII interface is a shell-like front end for the application. The network interfaces let tools communicate with other tools in the environment by multiplexing simultaneous network connections and implementing the Shastra communication protocol.[1]

The entire set of connected Shastra network interfaces implements the abstract Shastra layer at runtime (see Figure 1). This layer maintains the collaborative environment, provides access to different systems, and provides facilities for initiating, terminating, joining, leaving, and conducting collaborations.

Tools

Shastra tools are the building blocks of the runtime system. Kernels and session managers manage the distributed and collaborative environment. Shastra toolkits support scientific design and manipulation. Services provide mechanisms for communication and animation. We refer to toolkits and services collectively as front ends, or simply fronts, since they are the actual sites of user interaction. Any front can access the Shastra environment management tools to instantiate tools either locally or on remote sites, and to terminate previously instantiated tools. Fronts can connect directly to each other to exchange data in client-server or peer-peer settings using the Shastra layer.

Kernel. The Shastra kernel consists of a group of cooperating kernel processes. It maintains the runtime environment, tracking all instances of tools in the distributed system. A directory lets users dynamically discover what tools are active in the environment at any time. A location facility provides contact information about where the tools are running and lets applications connect to each other.

Session manager. A user starts a Shastra collaborative session through a front. One instance of a session manager runs per collaborative session. The session manager maintains a collaborative session, handles connection details, controls interaction, and regulates access. It is a repository of the shared objects in a collaboration, and it tracks membership of the collaborative group.

The session manager provides the multicast facility needed for information exchange in synchronous multiuser conferencing. A constraint management subsystem resolves conflicts arising from multiuser interaction, thus maintaining mutual consistency of operations. The session manager also provides a floor-control facility based on baton-passing.

Related work

Groupware refers to multiuser software that enables computer-supported cooperative work (CSCW). It focuses on using the computer to facilitate human interaction for problem solving. Ellis et al.[1] present an overview of the state of the art and identify issues in this area, which center on performing common tasks in a shared environment.

In the direction of shared environments, research in collocation has resulted in systems like Monet,[2] MMConf,[3] Rapport,[4] Collaborative Environment for Concurrent Engineering Design (CECED),[5] and Mermaid.[6] Some of these systems provide audio-video communication and content-independent sharing of drawing and viewing surfaces. Shastra lets us build applications with shared drawing and viewing surfaces by supporting content-dependent sharing—the applications are "collaboration aware" and support synchronous multiuser manipulations of application-specific objects. This adds a new dimension to collaborative problem solving, because it permits cooperative manipulation and browsing of objects and interaction in the context of applications that manipulate those objects. It supports cooperation in the design (problem-solving) phase as well as in the analysis (review) phase.

Colab is a collaborative meeting facility that lets multiple users manipulate drawn objects simultaneously.[7] Rendezvous presents a powerful architecture for multiuser applications and provides high-level support for building groupware in a distributed setting.[8] In its centralized architecture, all view generation is performed in a central server. This doesn't scale well in a graphics-intensive scientific setting. Shastra adopts a hybrid computation model alleviating problems of platform heterogeneity and performance.

In the scientific domain, Carlborn et al.[9] presented a teleconferencing approach to modeling and analysis of empirical data, and discussed a collaborative scientific visualization environment without output images of visualizations shared among multiple users. The Shastra environment makes it convenient to build collaborative visualization facilities that share not only the results of visualizations but also the input data and models. This sharing allows multiple users to interact over the data set while analyzing multiple simultaneous renderings with different viewing directions, cutaways, and independent visualization parameters.

Mercurio et al.[10] described an interactive visualization environment for 3D imaging, where an electron microscope is used as a computer peripheral. The Shastra environment promotes sharing of such unusual and expensive resources among multiple users across a network by enabling application-level cooperation.

The CSCW infrastructure of Shastra facilitates creation of collaborative multimedia applications. Shastra provides intuitive session initiation methods, flexible interaction modes, and dynamic access regulation.

References

1. C. Ellis, S. Gibbs, and G. Rein, "Groupware: Some Issues and Experiences," *Comm. ACM*, Vol. 34, No. 1, Jan. 1991, pp. 38-58.
2. K. Srinivas et al., "Monet: A Multimedia System for Conferencing and Application Sharing in Distributed Systems," *Proc. First Workshop on Enabling Technologies for Concurrent Engineering*, Concurrent Eng. Enabling Technologies Group, CERC, West Va. Univ., Morgantown, W.Va., 1992, pp. 21-37.
3. T. Crowley et al., "MMConf: An Infrastructure for Building Shared Multimedia Applications," *Proc. ACM CSCW 90*, ACM Press, New York, 1990, pp. 329-342.
4. S. Ahuja, J. Ensor, and D. Horn, "The Rapport Multimedia Conferencing System," *Proc. ACM SIGOIS*, ACM Press, New York, 1988, pp. 1-8.
5. E. Craighill et al., "Environments to Enable Informal Collaborative Design Processes," *Proc. First Workshop on Enabling Technologies for Concurrent Engineering*, Concurrent Eng. Enabling Technologies Group, CERC, West Va. Univ., Morgantown, W.Va., 1992, pp. 47-51.
6. K. Watabe et al., "A Distributed Multiparty Desktop Conferencing System," *Proc. ACM CSCW 90*, ACM Press, New York, 1990, pp. 27-38.
7. M. Stefik et al., "Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings," *Comm. ACM*, Vol. 30, No. 1, Jan. 1987, pp. 32-47.
8. J. Patterson et al., "Rendezvous: An Architecture for Synchronous MultiUser Applications," *Proc. ACM CSCW 90*, ACM Press, New York, 1990, pp. 317-328.
9. I. Carlbom et al., "Modeling and Analysis of Empirical Data in Collaborative Environments," *Comm. ACM*, Vol. 35, No. 6, June 1992, pp. 73-84.
10. P. Mercurio et al., "The Distributed Laboratory," *Comm. ACM*, Vol. 35, No. 6, June 1992, pp. 54-63.

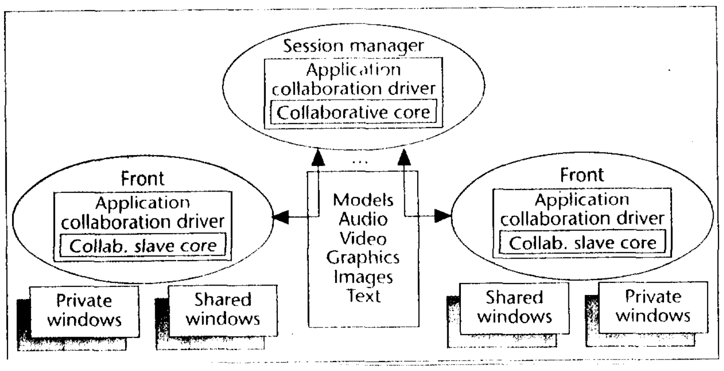


Figure 3. Architecture of a collaborative session. Fronts have separate private and shared workspaces. They maintain shared workspaces on behalf of the session manager. The substrate supports media-rich interaction between multiple fronts.

Figure 3 depicts the architecture of a typical collaborative session in Shastra. Figure 4 shows a view of the Shastra world, where different tools interact to support a collaborative environment. The Shastra collaboration architecture uses a replicated computation model for the multiuser system—a copy of the application (the front) runs at each site involved in the collaboration. The main benefits derived from this replication are support for hardware heterogeneity and good performance.

Toolkits. Scientific toolkits currently under the Shastra umbrella include Ganith, Shilp, Vaidak, Bhautik, Splinex, and Rasayan. These powerful stand-alone systems operate on application-specific models. We integrated them into the Shastra environment to permit concurrent engineering and distributed problem solving. Other toolkits can access these integrated toolkits. This interoperability enhances the functionality of each.

The Ganith algebraic geometry toolkit manipulates arbitrary degree polynomials and power series.[1] It is used to solve systems of algebraic equations and visualize the multiple solutions. Ganith incorporates techniques for multivariate interpolation and least-squares approximation to an arbitrary collection of points and curves, and C^1-smoothing using low-degree implicit patches. Other Shastra toolkits use the algebraic manipulation capability Ganith provides at its network interface—curve and surface intersection, interpolation, and approximation functionality.

Shilp is a boundary-representation-based geometric modeling system.[1] It provides extrude, revolve, and offset operations; edit operations on solids; pattern matching and replacement; Boolean set operations; fleshing of wireframes with smooth algebraic surface patches; and blending and rounding of solid corners and edges. Both local users and remote toolkits can invoke these operations.

Figure 5 shows a site during collaborative polyhedron smoothing in Shastra using Shilp and Ganith. Conferenced Shilp instances use multiple remote instances of Ganith to interpolate faces of

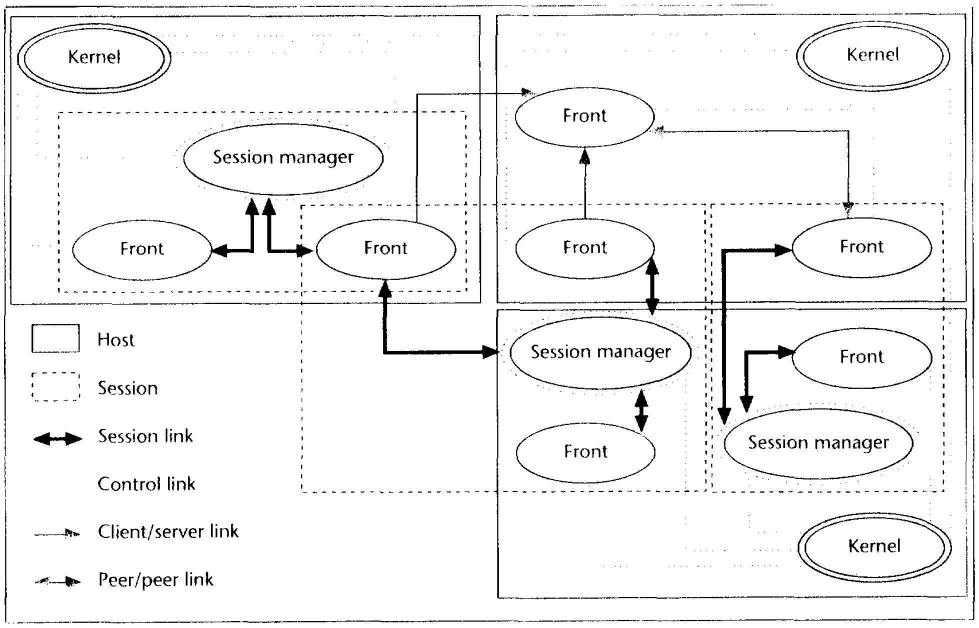


Figure 4. Information flow in the Shastra environment. Sessions can span many hosts, and fronts can participate in multiple independent collaborative sessions.

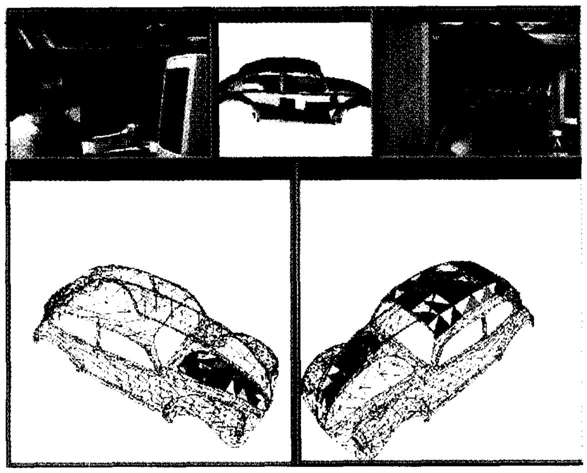


Figure 5. Collaborative polyhedron smoothing in Shastra. The session screen shows the original polyhedral car model (top center), one designer's part of the shared task (bottom left), and the shared, partially complete curved surface car model (bottom right). It also shows images of a supporting video conference.

a polyhedral car model in parallel and thereby produce a curved surface model with C^1-continuous surface patches. The toolkits communicate via their network interfaces.

The Vaidak medical image reconstruction toolkit constructs accurate cross-sectional, surface, and solid models of skeletal and soft tissue structures from computed tomography (CT), magnetic resonance imaging (MRI), or laser surface imaging (LSI) data. Shilp can use these models for design. Figure 6 shows a distributed problem-solving scenario in which a geometric model of a human femur is reconstructed in Vaidak and manipulated in Shilp.

The Bhautik physical analysis toolkit provides mesh generation facilities and a graphics interface to set up, perform, and visualize physical simulations on geometric models created interactively using Shilp or on models reconstructed in Vaidak from imaging data. Figure 7 shows a load-transfer finite element analysis for custom design of hip implants, using Vaidak, Shilp, and Bhautik.[1]

Splinex is a curve and surface modeling toolkit for interactive creation and manipulation of implicit and parametric splines in Bernstein-Bezier and A-spline bases.[1] It provides Bezier and A-spline surface manipulation capability in the Shastra environment.

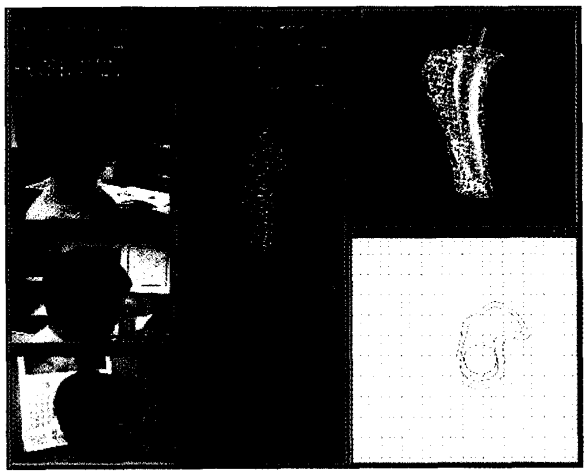


Figure 6. Collaborative customized hip implant design. A designer uses Shilp to interactively create a geometric model of a hip implant (top right) by generating cross-sectional contours of the implant (bottom center and right) from a sectional model of the femur (center) created in Vaidak. A video conference supports communication.

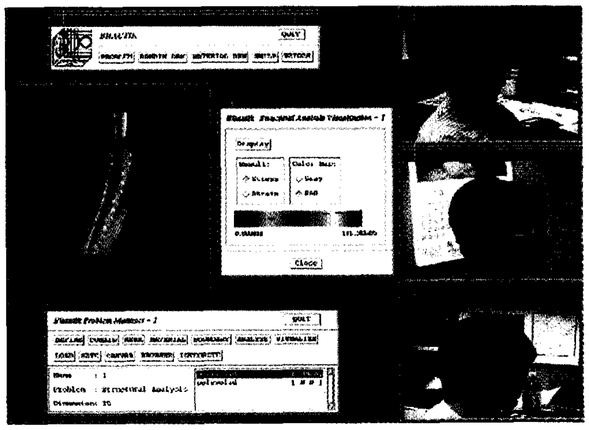


Figure 7. Stress analysis visualization. A designer uses Bhautik to analyze stress under loading patterns and thereby optimize the shape for a custom artificial implant for a human femur (top left). Video conferencing permits communication among designers.

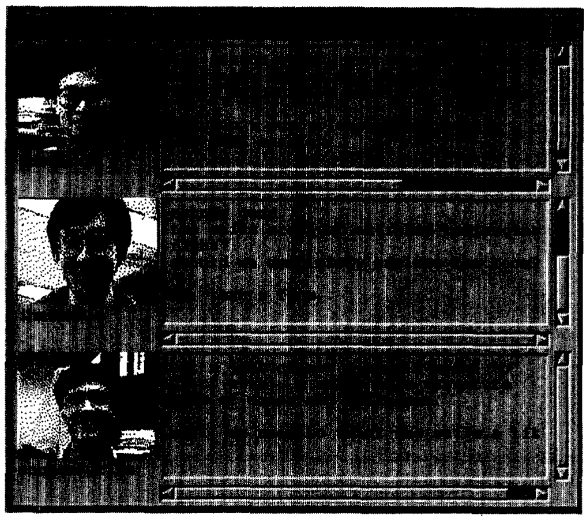


Figure 8. One site in a three-way text conference. Sha-Talk provides a simple textual conferencing facility. Image bitmaps identify the owner of a text panel.

Figure 9. 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). The images are generated by Vaidak from large volume data sets.

Rasayan can compute and visualize the "docking" of drug and protein molecules under molecular Brownian motion. It provides mechanisms for analysis and visualization of the potential energy surfaces of the molecules and the stationary points on these surfaces.

Services. The current set of Shastra services contains communication and animation tools. They provide a media-rich communication substrate for collaboration. In the design of multimedia applications, they relieve developers of the burden of low-level manipulation of devices and media formats. In engineering and scientific settings, especially in design and analysis, most information shared by a collaborating team is oriented towards structured 3D graphics. The information is typically application specific. However, including text, image, audio, and video communication greatly enhances the quality of interaction. In this section, we describe the services. In the next section, we present them in the context of a geometric design collaboration.

Sha-Talk is a text communication tool that supports synchronous *n*-way textual conversations. Shown in Figure 8, Sha-Talk is useful for designers who do not have multimedia communication facilities on the desktop.

Sha-Draw is a Shastra environment sketching tool for generating and displaying simple 2D pictures. It uses a rich set of primitive operations. A collaborative session with Sha-Draw lets a group synchronously create and edit simple 2D sketches on shared whiteboards. (A collaborative sketching session using Sha-Draw is discussed later in "The startup problem.")

Sha-Poly is a collaborative visualization and graphical-object browsing and manipulation tool. It supports shared viewing of 3D models using different display and visualization techniques in a synchronously conferenced setting. Figure 9 shows one of three sites with independent private windows and a shared conference window, using Sha-Poly for volume visualization of large medical data sets.

The Sha-Phone service records and plays back audio information stored in multimedia objects. An *n*-way audio conference is conducted by setting up a collaborative session consisting of Sha-Phone instances.

Sha-Video handles image data (without sound). The image data can be either still or motion video. It is used, both directly and by other tools, to play back and record video information stored in multimedia objects. A collaborative session with Sha-Video applications provides the mechanism to conduct a silent video conference. In Figure 10 a researcher uses a live video window to confirm the topological accuracy of a reconstructed femur in Vaidak.

Gati is an animation server that provides distributed and collaborative real-time interactive animation in two and three dimensions. The sys-

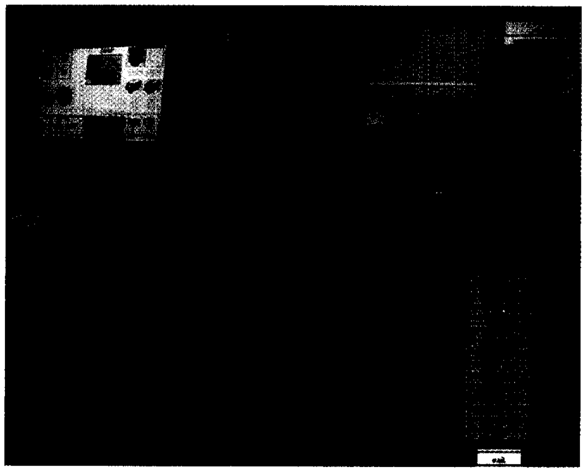


Figure 10. Video support for design. A researcher (top right) uses a live Sha-Video window (top left) to verify topological accuracy of a reconstructed femur model (bottom left, bottom right) in Vaidak.

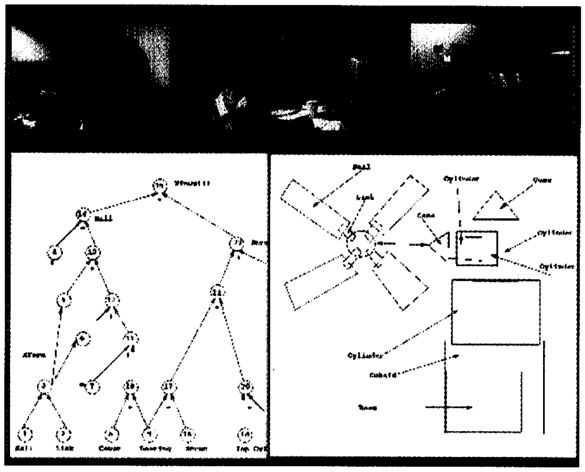


Figure 11. Using Sha-Draw for shared 2D sketching. Collaborating designers (top row) use shared windows to create a sketch (bottom right) and a design graph (bottom left).

tem supports a high-level animation language based on a commands/event paradigm.

Collaborative problem-solving scenario

We have built an application for collaborative design based on set operations. This approach to constructive solid geometry offers a flexible way to create intricate 3D models by performing set operations like union, intersection, and difference on simpler models. In solid modeling systems based on boundary representation, generating the results of set operations is computation intensive. However, this design process can be represented as a tree in which the lower levels are often parallelizable. This allows a group of designers to work independently on those parts. Our application improves design throughput by providing a collaborative environment from the conception phase through the final design.

The application uses two toolkits, Shilp and Sculpt. The design group creates a 3D model by using Shilp to perform set operations on models of increasing complexity and using Sculpt as a back end to perform the actual operations. Sculpt is optimized to perform set operations—union, intersection, difference, and complementation—on polyhedral geometric models.[2] The Shastra layer links the two toolkits and enables inter-application cooperation. User-level collaboration mechanisms result in a powerful multiuser interactive design facility.

The startup problem

One challenge of this design scenario is establishing a starting point. This issue is traditionally resolved by conducting a physical meeting to identify a design team, synchronize on a starting point, and agree on an idea of the final design. Shastra eliminates the need for such a physical meeting by providing mechanisms to create a design group and media-rich support for a design brainstorming session (see Figure 11).

A user running an instance of Shilp can query Shastra to find other active users in the environment. The user can then employ Shastra's messaging facilities to invite some of them to a text conference. Supported by Sha-Talk, potential members of the group can explore their interest in a particular design. The original user can decide who to include in the working group.

Before the design process starts, some group

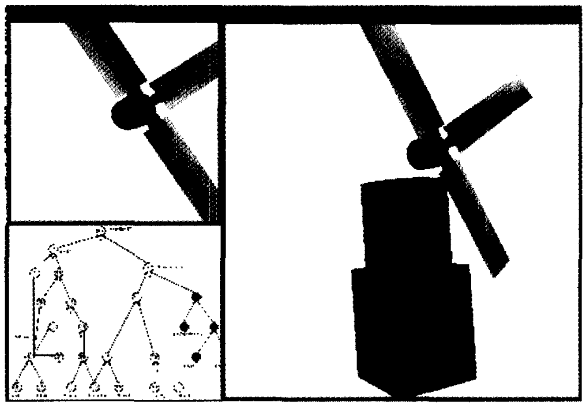


Figure 12. One site in a design collaboration. The design graph (bottom left) provides a context to monitor progress and regulate the task. The designer sees the incomplete shared model (right) and the locally designed part (top left).

members may have a mental picture of the object they are attempting to design, while others may not. One designer can initiate a collaborative brainstorming session using Sha-Draw. If audio and video processing hardware is available (as it is in Figure 11), the designer can invoke instances of Sha-Phone and Sha-Video, and initiate the relevant sessions. Sites without video hardware can use the software-only playback facility to display incoming video streams and transmit pre-recorded outgoing streams. Through this audio-visual communication, the group can rapidly establish the design goal. Then they can interactively create a rough sketch of the intended design. Alternatively, Sha-Video can broadcast a stored or live image of an actual physical object (or its picture) to the group. At the end of this phase, the entire team has a good idea of the task at hand.

The designers use Sha-Draw to set up the dependencies of the various parts of the design in graphical form. They create a directed design graph (see Figure 11), where nodes indicate solid models and edges indicate dependencies of the destination nodes. The leaf nodes (that is, nodes that are not dependent on other nodes) represent existing or primitive solid models, and internal nodes represent intermediate models in the design process. A designated root node represents the final design goal. Directed edges indicate that the destination node results from an operation on all of the source nodes. Annotations in the graph indicate the operation required to obtain the destination node from the source nodes.

Design outline

The design outline occurs in two phases, design graph generation and model computation. The design graph, created in a Sha-Draw collaboration referring to a sketch or video image of the final model, is a succinct summary of the entire design task. Designers can store the image and/or sketch with the graph for future reference. Shilp converts the graph into a form amenable to this operation, with maximum in-degree of the nodes being 2 (because Sculpt supports only unary and binary operations). An automatic disjunctive normal form (DNF) decomposition is the simplest transformation, but it doesn't produce the most efficient design graph. The design team can cooperatively restructure the design graph in a Sha-Draw collaboration.

In the model computation phase, a designer graphically positions models through the Shilp user interface. This sets up models in appropriate configurations for set operations. The actual operations to generate intermediate and final models of the design graph are performed in Shilp by automatically requesting geometric services from Sculpt.

Design process in the Shastra setting

In a single-user setting, a designer would compute the various graph nodes sequentially. The final model would be checked for goodness, and the computation process repeated until a satisfactory model resulted.

In the Shastra multiuser setting, a Shilp user initiates a collaborative session. This user tells the local kernel which other users are invited to participate in the session, and (by default) becomes the group leader. The kernel instantiates a session manager, which starts a session with the group leader as its sole participant. Then it invites the specified users of concurrently executing remote Shilp instances to participate. Users who accept are incorporated into the session. The session manager provides access to shared objects and context at all participating sites.

Any participant can leave the session at any time by simply unlinking from it. Users of other instances of Shilp in the environment can query the system to discover ongoing sessions and request participation. The group leader regulates whether or not they are allowed to join the session. The leader can also invite other Shilp users to participate in the session.

Every participating Shilp session creates two shared windows in which all the cooperative interaction occurs. (More local windows can be created if desired.) One shared window contains the design graph used to regulate the entire operation, and the second window contains models as they are created or introduced to the session (see Figure 11). Users introduce leaf node objects into the session by selecting them into the shared window.

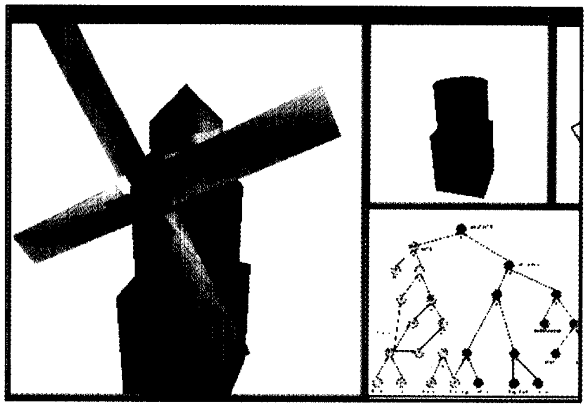


Figure 13. Another site, at the end of a collaborative design session. The designer sees a completed shared model (left), the locally designed part (top right), and the shared design graph (bottom right).

The session manager partitions the design graph into slightly overlapping zones. It bases the partitioning on the number of people in the collaborating group and on the number of subtasks left in the operation (the number of uncomputed nodes in this case). It aims to minimize the number of shared partition nodes. Shared nodes in the graph are regions of contention in this collaboration scenario, since they constitute dependencies in an otherwise parallelizable situation.

The partitioning also aims to distribute the leaf nodes equally among the designers, since they usually represent nodes that must be created interactively. The partitioning defines a scenario for fair, minimal conflict, cooperative interaction. It is dynamically altered as users join and leave the session. The group leader can explicitly specify and alter the partitioning.

Shastra displays the partitioned design graph in a shared window, which provides a context to regulate the collaborative operation, since it captures the state of the operation. The system assigns partitioned zones to the collaborating designers and colors each one differently for identification. Every user is responsible for filling the intermediate nodes in his or her zone by first positioning the models on the incoming edges and subsequently performing the actual set operation. This process repeats until a satisfactory design results. Figure 12 shows one site in the design of a simple windmill model. Figure 13 shows another site at the end of the operation.

Collaborative interaction

The session manager regulates all interaction relevant to the operation at participating sites. Interaction can occur in two modes. In the regulated mode, the user responsible for a zone creates all the models internal to the zone. The session manager denies all other users access to the zone interior. If the source nodes for an intermediate node are filled, a user locks the intermediate node by selecting it in the design graph window. The session manager allows this user access to models in the source nodes. The user interactively positions these models and performs the appropriate set operation. The resulting model is assigned to the intermediate node, which is subsequently unlocked.

Users of adjacent zones must agree about the models at the boundary node to avoid inconsistencies in the design. A good group design protocol for this setting is to resolve boundary condition issues at the start of the operation. This prevents unnecessary cycles later on caused by inconsistencies. To implement this protocol, the users must compute the subgraphs rooted at boundary nodes first, with no further design activity proceeding until they obtain satisfactory models at those nodes.

All operations are performed via the central session manager, which keeps all sites up to date. Thus, users have a dynamically changing view of the operation in the shared windows—the design graph and intermediate models. Changing a node requires all of its dependencies to be reevaluated. The operation is completed when all the design graph nodes have been evaluated. Any site with copy permission can then extract the model from the session and save it.

Access regulation and collaboration modes

Shastra's collaboration infrastructure supports a two-tiered permissions-based access regulation mechanism. This mechanism structures a variety of multiuser interaction modes at runtime. It allows a high degree of tailoring and flexibility in CSCW applications, both in interaction and in data sharing and access control.

The regulatory subsystem supports access, browse, modify, copy, and grant permissions for collaboration sites as well as for shared objects.[1]

These permissions control what actions different users in the conference can take and what objects they can operate on. In addition, tools can define and use new permissions for tool-specific actions. The group leader controls permissions.

The communication facilities permit rapid exchange of rationales for design, analysis, and iterative shape modification.

The regulatory subsystem also provides a mechanism to enforce and regulate floor control based on turn-taking. Users can dynamically configure the interaction mode and permissions to suit the task. In the brainstorming phase, for example, it is useful to allow everyone equal access to all operations and objects. This supports the free flow of ideas.

In the unregulated mode for this operation, the partitioning merely suggests a minimal-conflict setting, and the session manager doesn't regulate interaction beyond what is specified by collaboration permission settings for the site and object. In this mode, users can access any node if they have access and modify permissions for the collaboration.

The session manager lets only one user manipulate a "hot spot" in the graph—where there is a possibility of contention—at any particular instant. It uses the first-come-first-served paradigm to decide which user gets temporary exclusive control. The last completed operation specifies the model associated with the node.

The system's baton-passing facility can be used for floor control—to take turns to set boundary nodes. Alternatively, designers can use the auxiliary communication channels to regulate access and to decide which users will set those nodes.

At one extreme, a single designer can use the Shastra implementation much like a noncollaborative environment. Allowing other users to join the session with only access and browse permissions sets up the environment like an electronic blackboard. In this way, novice users could learn the basics of the design mechanism. An appropriate setting of collaboration permissions and turn-taking might allow hands-on experience with the task. In conjunction with Shastra's audio and video communication services, this becomes a powerful learning environment.

In a different situation, a group of designers can set up a regulated collaborative session and collaborate to design an object. Each designer performs only the designated part of the shared design. This can speed up problem solving by as many times as the problem can be partitioned. Novice designers can join the on-going session with only access and browse permissions and thus become familiar with the group dynamics of a collaborative session.

In yet another situation, a group of designers can start an unregulated collaborative design session. Judicious use of the auxiliary communication facilities (audio, video, and text) to regulate design operations can speed up the process by a factor equal to the number of participants.

Heterogeneity issues

A Shastra conference consists of multiple tool instances at different sites. This localizes platform-specific dependencies in the tool. It lets the session manager view tools as high-level application objects, without concern for details of how things are actually done. This approach supports the Shastra environment on a variety of hardware platforms. Specifically, tools can take advantage of graphics, video compression and decompression, and audio processing hardware available in the environment. Thus, the Shastra architecture greatly simplifies multimedia interaction management.

The application scenario we described above is a *homogeneous collaboration*. In other words, the collaborative design task is supported on a collection of instances of the same tool (Shilp in this case). We are currently building an environment for collaborative design of custom hip and knee implants using different Shastra toolkits coupled with a computer-aided manufacturing facility. This puts us in the realm of heterogeneous collaborations between instances of different tools, which operate on different types of models or data.

Shastra's architecture facilitates the kind of interapplication cooperation required to build such a system. The design team uses Vaidak to build a model of the patient's femur from cross-section information (CT/MRI images) and Shilp to custom design an implant for the femur (see Figure 6). A physical analyst uses Bhautik to conduct a stress-strain analysis evaluating the load transfer between the implant and bone (see Figure 7). The design team iterates this custom design process until they obtain an optimally shaped customized implant. The multimedia communication facilities of Sha-Video, Sha-Phone, and Sha-Poly conferences permit a rapid exchange of rationales for design choices, interpretations of analyses, and iterative shape modification and analysis.

Conclusions

Shastra is a powerful distributed and collaborative environment. The Shastra layer provides an enabling infrastructure for rapid prototyping of tools. The runtime environment helps to build multiuser applications. Though we have focused our implementation on scientific and engineering design, Shastra easily abstracts to other situations. The Shastra layer is generic and can be used to implement systems for collaborative editing, code viewing and quality assurance, software development, CAD applications, and even interactive multiplayer games. **MM**

Acknowledgments

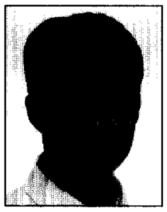
We thank the anonymous referees for their comments and suggestions.

A preliminary version of this article, titled "Collaborative Multimedia Scientific Design in Shastra," appeared in the *Proceedings of the First ACM International Conference on Multimedia*, ACM Press, New York, 1993, pp. 447-456.

This work was supported in part by NSF grants CCR 92-22467, DMS 91-01424, AFOSR grant F49620-93-10138, NASA grant NAG-1-1473, and a gift from AT&T.

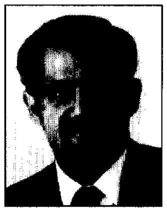
References

1. V. Anupam and C. Bajaj, "Shastra: An Architecture for Development of Collaborative Applications," *Proc. Second Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises*, IEEE Computer Society Press, Los Alamitos, Calif., 1993, pp. 155-166.
2. W. Thibault and B. Naylor, "Set Operations on Polyhedra Using Binary Space Partitioning Trees," *Computer Graphics (Proc. Siggraph)*, Vol. 21, No. 4, 1987, pp. 153-161.



Vinod Anupam is a doctoral candidate in the Department of Computer Sciences at Purdue University, West Lafayette, Indiana. His research interests include computer-supported cooperative work and groupware, networking and distributed systems, geometric modeling, graphics and visualization, hypermedia, and graphical user interfaces.

Anupam received his bachelor's degree in computer science from Birla Institute of Technology and Science, Pilani, India in 1988. He is a member of Upsilon Pi Epsilon.



Chandrajit L. Bajaj is a professor in the Department of Computer Sciences at Purdue University, West Lafayette, Indiana, and directs the Collaborative Modeling and Visualization Laboratory. His research is in computational science, geometric modeling, computer graphics, scientific visualization, and distributed and collaborative multimedia systems.

Bajaj graduated from the Indian Institute of Technology, Delhi in 1980 with a bachelor's degree in electrical engineering. He received his MS and PhD in computer science from Cornell University, Ithaca, New York in 1983 and 1984, respectively.

Readers may contact Bajaj at the Dept. of Computer Science, Purdue Univ., West Lafayette, IN 47907, e-mail bajaj@cs.purdue.edu.

More information on Shastra software is available via anonymous ftp from ftp.cs.purdue.edu and via the World Wide Web server using xmosaic from http://www.cs.purdue.edu/research/shastra/shastra.html.