MULTI-USER MODELING OF NURBS-BASED OBJECTS

Cheryl V. Foster  Yuri Shapirstein  Christopher D. Cera  William C. Regli
Geometric and Intelligent Computing Laboratory
Department of Mathematics and Computer Science
Drexel University
3141 Chestnut Street
Philadelphia, PA 19104
http://gicl.mcs.drexel.edu/

ABSTRACT

This paper presents MUG, a multi-user environment for collaborative conceptual shape design. The majority of current research prototypes and commercial systems for collaborative modeling emphasize data sharing and markup. Collaborative 3D environments and virtual worlds usually restrict themselves to models with display representations (e.g., VRML) and asynchronous sharing of distinct objects. Our MUG prototype enables collaborative modeling of individual shapes: multiple users, operating at a distance, working on the same design. In our current work, we have focused on collaborative shape modeling of NURBS—allowing users to select control points and concurrently manipulate the same entity.

To achieve this functionality, we have developed a protocol for synchronous mathematical construction of NURBS forms. We have adopted a multi-client/single server architecture in which changes are synchronized at a collaboration server and propagated to the clients. Our implementation uses Sun Microsystems’ Java, along with the Java3D and JavaSpaces extensions. In addition, as part of this work, we have developed a Java-based NURBS library, as well as faceting and rendering routines specifically suited for a light-weight collaborative environment.

We believe that as network connectivity continues to radically reshape the design process, we need to enable new design techniques based on intimately shared design spaces. We believe that our approach illustrates one channel in which collaborative engineering systems can be better integrated with collaborative work environments.

INTRODUCTION

This paper presents an approach to collaborative modeling and conceptual shape design. The majority of current research prototypes and commercial systems for collaborative modeling emphasize data sharing and markup; collaborative 3D environments and virtual worlds usually restrict themselves to simple models and asynchronous sharing of distinct objects.

We are creating an environment that allows a team of designers to operate simultaneously on a single, shared shape object. MUG, short for Multi-User Groups for Conceptual Understanding and Prototyping, is our research platform. MUG allows teams of users to operate in a multi-user 3D virtual workspace to create and annotate conceptual designs of electro-mechanical devices.

To achieve this functionality, we have developed a protocol for synchronous mathematical construction of NURBS forms. We have adopted a multi-client/single server architecture in which changes are synchronized at a collaboration server and propagated to the clients. Our implementation uses Sun Microsystems’ Java environment, along with the Java3D and JavaSpaces extensions. In addition, we have developed a Java-based NURBS library, as well as faceting and rendering routines specifically suited for a light-weight collaborative environment.

While network connectivity reshapes the design process, we
need to enable new design techniques based on intimately shared design spaces. We believe that the approach we illustrate with MUG is one channel in which collaborative engineering systems can be better integrated with collaborative work environments.

This paper highlights the collaborative and free-form features of MUG, just one of the research thrusts being explored. The collaboration and free-form features each bring to MUG more robust functionality over the previous versions of the application. Presented together, they demonstrate the users ability to simultaneously manipulate different portions of a NURBS form via control points, emphasizing the concurrency issues that can arise when working on a complex design.

This paper is organized as follows: Background and Related Work gives an overview of related research work from a variety of research fields. The Technical Approach section describes our approach to collaborative modeling and details the implementation technology we used to deploy our prototype. Working with MUG gives a walk through of our system during a multi-user modeling session. Discussion and Conclusions presents our conclusions and discusses open areas for our future work.

BACKGROUND AND RELATED WORK

Surface Representation. NURBS (Non-Uniform Rational B-Splines) is the standard representation for surfaces and shapes in most commercial CAD packages available and has been incorporated as part of IGES and PDES/STEP data transfer standards (Au and Yuen, 1995). Piegl (Piegl, 1991) attributes the popularity of NURBS to several factors:

- Both analytic and free-form shapes can be represented precisely since they have a common mathematical form that can represent both.
- They provide the means to design a multitude of shapes thru the manipulation of control point weight and position.
- They are invariant under normal transformations (scaling, rotation, translation, shear) as well as parallel and perspective projections.
- Computations are stable and relatively fast.
- They have clear geometric interpretations and have a powerful tool kit.

NURBS curves are defined as a piecewise rational polynomial function with the following form:

$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u)w_i P_i}{\sum_{i=0}^{n} N_{i,p}(u)w_i} \quad \text{where } a \leq u \leq b$$  \hspace{1cm} (1)

where the $P_i$ are the control points, the $w_i$ are weights associated with the control points, and the $N_{i,j}$ are recursively defined as:

$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u < u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,j}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,j-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,j-1}(u)$$ \hspace{1cm} (2)

On the knot vector:

$$U = \{a, \ldots, a, u_{p+1}, \ldots, u_{m-p}, b, \ldots, b\}$$

$$V = \{0, \ldots, 0, u_{q+1}, \ldots, u_{s-q}, 1, \ldots, 1\}$$

There are several different types of NURBS surfaces that can be defined, such as ruled, extruded surfaces, or surfaces of revolution, which are based off of either one or two curves. In general they are defined by equation 3,

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}P_{i,j} \sum_{i=0}^{n} \sum_{j=0}^{m} N_{i,p}(u)N_{j,q}(v)w_{i,j}$$  \hspace{1cm} (3)

with $P_{i,j}$ forming a control net of points, $w_{i,j}$ are the weights, $N_{i,p}(u)$ and $N_{j,q}(v)$ defined on the knot vectors

$$U = \{0, \ldots, 0, u_{p+1}, \ldots, u_{m-p}, 1, \ldots, 1\}$$

$$V = \{0, \ldots, 0, u_{q+1}, \ldots, u_{s-q}, 1, \ldots, 1\}$$

where $r = n + p + 1$ and $s = m + q + 1$. (Piegl and Tiller, 1997; Piegl, 1991)

Interactive Modeling. In developing MUG, we looked at a number of existing techniques for interactive shape and solid modeling, including virtual clay modeling, free-form deformations, as well as techniques for modeling in immersive and semi-immersive virtual environments. A considerable amount of this research comes from the computer graphics community, who are interested in building realistic virtual environments.

Free-form deformations (Lamousin and Jr., 1994), and more specifically NURBS-based free-form deformations, create an object by conceptually embedding it in a "clear" pliable solid. Manipulations are applied to the surrounding solid and carried over.
through to the encased object. This prevents the object from being directly manipulated by the user and reduces the problem of hidden control points that may define the embedded objects.

Similar to free-form deformations, virtual clay modeling (Kameyama, 1997; Krause and Liddemann, 1996) is an interactive modeling technique that also provides a way of indirectly manipulating an object. This is done via virtual tools such as scrapers, templates, and true sweeps. These systems let the user create more complicated designs than a simple interface compared to what is possible via free-form deformations alone. This simpler interface alleviates the problem of the user having to learn the complicated commands and/or operations of conventional 3D CAD Systems.

Work has also been done on shape modeling in virtual environments. An example of this is COVIRDS: Conceptual Virtual Design System (Dani et al., 1997) by Dani et al. at the L-CARVE laboratory in the Mechanical Engineering Department of the University of Wisconsin-Madison. By providing a simplified interface akin to virtual clay modeling, this system overcomes the difficulties of typical CAD systems. Instead of providing a piece of clay to be modeled, CORVIDS is a virtual reality system in which the user is able to give general commands verbally or grab and manipulate the object within the virtual environment.

Along the lines of COVIRDS and modeling in virtual environments, research is also being done on 3D sketching of free-form curves and surfaces. Stork et al. (Stork et al., 2000) have developed a method of "sketching" free-form surfaces based on sketched curves using a Virtual Table, stereo-glasses, transparent pad, and pen. In this system the user is able to draw multiple curves and create a free-form surface based on it.

Collaborative Work Environments. These include traditional Computer-Supported Co-operative Work (CSCW) (Grudin and Poltrock, 1997) systems, as well as those enabling shared 3D. At Stanford (Frohlich et al., 1997; Kruger et al., 1995), collaborative production modeling and planning has been performed using the Responsive Workbench, an environment in which the user is able to directly explore and maneuver the objects in the environment, and to relate with other participants as with a tactile virtual reality space. Much of the research in this area has focused on information requirements for synchronizing shared realities (Pang and Wittenbrink, 1997) and managing networked bandwidth requirements (Diase et al., 1996; Lamotte et al., 1996).

Computer-Aided Conceptual Design. There has been a realization in the CAD community that digital support for conceptual design is an increasingly critical component of software environments for designers (Sturgis et al., 1993). Current research on Computer-Aided Conceptual Design (CADC) has produced several prototype systems. Wallace and Jakiela (Wallace and Jakiela, 1993) developed an experimental conceptual design tool for industrial designers capable of generating alternative design suggestions based on user input. The purpose of the tool is to provide a means for designers to quickly generate and adapt alternate design concepts. Research also included systems for "conceptual sketching" (Hearst et al., 1998) in Human-Computer Interface design as well as CAD (Eggl et al., 1995).

Also relevant to our work is the CONGEN system of Gorti and Sriram (Gorti and Sriram, 1996; Gorti et al., 1998). Their work presented a design framework for symbol-to-form mapping, which consists of deriving spatial relationships between objects as a consequence of the functional relationships, instantiating alternative feasible solutions subject to these relationships, and presenting the evolving descriptions of geometry.

Collaborative Virtual Worlds. Some of the existing work on collaborative computer-aided design involves the use of immersive virtual reality systems. This work includes efforts at Stanford (Frohlich et al., 1997; Kruger et al., 1995) on collaborative production modeling and planning using the Responsive Workbench; work at University of Illinois at Urbana Champaign on vehicle design using a CAVE; and work at the University of Wisconsin-Madison (Arangarasan and Gadh, 2000) where they are developing a Detailed Virtual Design System for collaborative design in a multi-modal virtual environment. In these environments, the user is able to directly explore and maneuver around the objects in the environment, as well as to relate with other participants with a tactile virtual reality space. These systems require high bandwidth networks and considerable amount of custom graphics hardware (Dias et al., 1996; Lamotte et al., 1996).

Research on object selection has included the ability of a user to select an object while in an immersive environment. In Bowman (Bowman and Hodges, 1997) the authors describe a combination of techniques such as the 'arm-extension' and 'ray-casting' paradigm with which the user is able to select and manipulate objects. Research on collaborative object manipulation has also been studied by Pang and Wittenbrink in a system called Cspray (Pang and Wittenbrink, 1997). The authors created a protocol by which users may request/release control of objects, users may share or exchange views, and may modify or may lock objects. They also focus on related issues such as network latency and three dimensional pointers in a collaborative space.

Other research issues have included how to handle multiple pointers in an environment to manipulate objects in a more intuitive and efficient manner. It was shown that users using multiple simultaneous input perform better than with a single input mode (Zeleznik et al., 1997). This was also taken in consideration as the prototype was developed; however, due to environment restrictions, different approaches were taken, though they rely on much of the same intuition as multiple pointer manipulation.
TECHNICAL APPROACH

We view conceptual design as an iterative process in which a team of designers simultaneously refine a semantic and shape description of an artifact. Our approach integrates ideas from the different disciplines mentioned above, mixing shape modeling and Virtual Reality with semantic markup. The remainder of this paper will describe how to utilize NURBS as the basis for surface representation in a multi-user collaborative design space.

Implementation Overview

JavaSpaces. JavaSpaces (Freeman et al., 1999) is the Java based implementation of a “Tuple-Space” paradigm. Linda (Carriero and Gelernter, 1991), one of the best known implementations of this paradigm, was created at Yale University in early 80’s. Linda featured distributed architecture allowing to place different pieces of the system on different machines. The current release of JavaSpaces is bound to a single machine, but is built on top of Jini Java’s seamless discovery system.

As seen in figure 1, clients perform operations that map entries or templates onto JavaSpaces services. These can be singleton operations (as with the upper client), or contained in transactions (as with the lower client) so that all or none of the operations take place. A single client can interact with as many spaces as it needs to. Identities are accessed from the security subsystem and passed as parameters to method invocations. Notifications go to event catchers, which may be clients themselves or proxies for a client (such as a store-and-forward mailbox).

Tuple-Space paradigm features a system for storage of serialized objects, that can be retrieved by any client connected to the Tuple-Space, as all of the objects are considered to be public. Objects stored in the space can be read or taken (i.e. removed), and modification is done by first removing an object, modifying it locally, and placing it back into the space. The retrieval of the objects is done using pattern matching, and out of a group of objects that fit the pattern, one random object is selected.

JavaSpaces was picked to handle data transfer for its ease of use and a set of tools that it provides. As was mentioned above, JavaSpaces guarantees complete synchronization which removes responsibility from the developer to provide such capability. The next feature that attracted our attention was its random data access compared to sequential access, using powerful pattern matching techniques to prioritize messages. Another important aspect is that JavaSpaces transfers data using Java objects instead of byte streams. Thus there is no need to worry about message parsing and completeness.

Messages that are found in the JavaSpace during the ongoing session, can be partitioned into two categories: short-life and long-life messages. Short-life messages are the ones passed from client to server or agent or the other way around. Long-life messages are created by the agent and represent changes done to a particular design. All messages are similar to KQML (Knowledge Query and Manipulation Language) (Finin et al., 1995) (they contain a subset of properties that KQML does, but unlike KQML they are not text based but object based).

Java3D. There are four different groups of visualization software development kits (SDKs) available for use with the Java language and development environment: Java-vrml, Java3D, Java/OGL, and other third party SDKs. Java-vrml was set aside due to the difficulty of coding and inability to extend its widely used format. Both Java/OGL and other third-party products were rejected because of lack of support. Java3D, on the other hand, is a flexible technology which seems to have a bright future, and is easily extensible since there are no fixed boundaries set.

MUG: History and General Features

MUG is the third iteration of a tool developed at GICL for conceptual design. Its predecessor, CUP (Conceptual Understanding and Prototyping) (Anthony et al., 2001; Shapirshteyn et al., 2000), began as a Netscape-based JavaScript-plus-VRML conceptual CAD application developed prior to the release of the first version of the Java3D API. In the second release, CUP evolved into a stand-alone Java3D based tool adding features such as load/save using XML (Consortium, 2000) and importing VRML files.

What makes this newest version of the application different is its ability to collaborate over the Internet along with other designers in real time. Unlike its predecessors, MUG also includes an extensive API allowing for plugins of different kinds such as VRML imports and NURBS extensions (as seen in Figure 2).
MUG System Architecture

There are three major components that make up MUG’s multi-user architecture: MUGServer, MUGClient and a JavaSpaces enabled system. Even though MUG has no control over JavaSpaces, JavaSpaces has to be initialized with a proper set of parameters that suit MUG’s needs. The overall architecture is shown in Figure 3.

**Server.** The Server has two pieces to it: MUGManager (see Figure 4 (a)) and Design Agent (see Figure 4 (b)). MUGManager is most important component of the system, it knows about all the users present during the collaboration, and which design agent (design session) each user is associated with. All of this information is also made available to each user, as we believe this to be a friendly work environment.

The Design Agent is an entity that is started by the server when such need arises (i.e. a user makes a request for a new design space). While there could be multiple agents present at one time, Figure 4 (b) shows them as one entity. Currently, a design agent is a part of the same Java Runtime Environment (JRE). In the future, however, might exist on a different machine altogether. Each agent is associated with a separate design and is responsible for tracking down messages describing changes done to its particular design. It then marks the change with its sequence number and encodes it into a slightly different format, which unlike other messages is not removed from the space, but remains there for all clients working on this design to pick up.

**MUGClient.** Unlike the components described previously, the client has a much more complex architecture (as seen in Figure 5). It features two threads of control, as well as a number of

![Figure 2. NURBS IN MUG: (a) A 7 CONTROL POINT NURBS CURVE; (b) A 7x7 CONTROL POINT NURBS SURFACE.](image)

![Figure 3. SYSTEM ARCHITECTURE OF MUG.](image)
The **BehaviorManager** binds the user actions (mouse events) to the software that generates proper responses. An example of this would be the fact that you get a component-associated GUI along side of the viewing canvas, which helps you make precise changes to the selected object.

The **PluginLoader** is the latest addition to the core of MUG. It allows MUG to hook additional components to MUG that were written using MUG’s API. This paper is centered on the workings of one of such components that allows for the drawing and manipulation of NURBS.

### Basic NURBS Manipulation in MUG

#### Drawing NURBS.

The free-form curves and surfaces in MUG are represented with NURBS, the standard representation in CAD packages. The shape of a NURBS form is dictated or controlled by four main parameters: the *control points*, *weights*, the *degree* of the curve, and the *knot vector*, (i.e. degrees of freedom).

In the current version of MUG, objects are not explicitly created. Users select pre-defined objects such as Box, Sphere, Curve, and Surface to add to the design they are working on. For simplicity, curves are pre-defined by setting a fixed number of equally spaced control points, all weights set to 1, the knot vector set to \{0, \ldots, 1\}, and the degree equal to 4. Surfaces are defined based on the control points of two 4 degree curves, one created in the \(x - y\) plane, the curve \(C(v)\), and the curve \(C(u)\) created in the \(y - z\) plane. The additional points that determine the control net are computed based on the intersection of curves identical to \(C(u)\) and \(C(v)\) from each control point in the other curve. For example: \(C(u)\) has control points \(P_u\) (represented as an array \(\mathcal{P}\{\mathcal{V}\}\)) and \(C(v)\) has control points \(P_v\) (\(\mathcal{P}\{\mathcal{V}\}\)), where the first element in \(P_u\) equals the first element in \(P_v\), the

---

*Figure 5. MUGClient ARCHITECTURE.*

---

*Figure 4. INTERNALS OF THE MUG SERVER-SIDE*
2 dimensional array P[i][j] that represents the control net is determined with the following pseudo code:

\[ P[i][0] = P_V[i] \]
\[ P[0][j] = PJ[j] \]
\[ P[i][j].x = P[i-1][j].x + (P[i][0].x - P[i-1][0].x) \]
\[ P[i][j].y = P[i-1][j-1].y + (P[0][j].y - P[0][j-1].y) \]
\[ P[i][j].z = P[i][j-1].z + (P[0][j].z - P[0][j-1].z) \]

where \( i = 0 \ldots \text{(length of } P_V[i]\text{)}-1 \) and \( j = 0 \ldots \text{(length of } PJ[j]\text{)}-1 \) The weights for all of the control points are equal to 1 and the knot vectors for the two determining curves are both \([0, \ldots, 1]\).

They were implemented with the Java3D extension to the Java programming language. Java3D does not have direct support for NURBS, so we created our own library of classes that utilizes the data types and rendering routines to create, store, and draw NURBS curves and surfaces. Algorithms for calculating the NURBS are based on the Equations 1–3 found in (Peg and Tiller, 1997).

Manipulating NURBS. In order to take advantage of the built-in object manipulation capabilities of MUG, we have implemented a limited degrees of freedom approach. For simplification purposes, a NURBS surface can only be manipulated thru the changes to the control points and their weights, while the degree of a curve can also be modified. This reduces the ambiguity and difficulty that arises when using this approach, but also limits the forms that are possible.

The NURBS library was implemented on the MUG API that we developed, designed to be a plug-in to MUG. A ControlPoint object was created and is associated with a particular NURBS form. Whenever a ControlPoint object is changed, it calls the appropriate functions for the NURBS form. For example when a control point on the screen is moved, the control point object is updated and that in turn updates the NURBS via the setControlPoint() function. This updates the control point value in the NURBS form, which is then completely recomputed and redrawn. Similarly for when the weight of the control point is changed.

The NURBS plugin takes advantage of the MUG API, by allowing it to manage the events that occur on the object while it does the computations and creates the Java3D Shapes. This is how the appearance of real-time updates are achieved. The selection and manipulation through the mouse of a control point generates many events, and changes to the NURBS form. Each time the curve or surface is recomputed, but only the overall change to the point is registered to the UndoManager, signaled by release of the mouse button.

**Multi-User Manipulation of NURBS**

Even though JavaSpaces guarantees total synchronization of objects stored in the space, it does not guarantee the sequence in which objects get selected if a pattern matches more than one item. For this particular reason a ticketing system has been invoked. Every time a change has to be submitted, the client picks up a ticket and submits it along with the change to the agent. Therefore no two changes can be processed at the same time.

Although it is up to the developer to decide how changes are to be stored, changes that are applied to NURBS carry a notion of globallity (e.g., position changes reflect a global position, not a local shift). Now assuming that two users have made a position change to the same control point, we know the first change will be canceled by the second one.

One of the elements of multi-user world that is left out is user coordination. If one user is manipulating object 1, and the second user is manipulating object 1, then only the changes made by the second user will be implemented. This is done for 2 reasons: first, network latency will not allow us for instantaneous notification, second, different properties of an object are treated separately, that means that either the whole object has to be frozen during the modification of either of its properties or each separate property has to be frozen. If the first approach is taken, this makes manipulation of NURBS objects by multiple people is impossible, the second approach, on the other hand, will flood the design space with flags.

**MUG Design Issues: Analysis of Communication and Computational Complexity Tradeoffs**

In the design and implementation of MUG, there were trade-offs between network bandwidth utilization and computation. One approach saves computer cycles on the MUGClient at the expense of client memory and network bandwidth utilization. The other approach saves memory and bandwidth, but incurs computational costs since NURBS objects are re-calculated as changes are propagated. The major parameters for sharing NURBS surfaces are:

- \(|P_u|\) and \(|P_v|\): the number of control points for the two defining curves, \(U\) and \(V\);
- \(g_u\) and \(g_v\), the granularity of the mesh in the \(u\) and \(v\) directions, respectively.

Using the entities described previously, we can assess the data storage requirements for memory and for transfer over networks, as well as the costs for the computations.

The following is the estimation for the first approach: Given the mesh granularity \(g_u\) and \(g_v\), the total number of nodes that define the surface is \(g_u \times g_v\). Each node is described by a 24 byte array, that makes up 3 double precision numbers (the \(x - y - z\) coordinates).
coordinates of the mesh point). The number of triangular facets that make up the surface is \((4(g_a-1)(g_r-1))\). Each triangle is defined by 3 indices which are 4-byte integers each, and 1 normal vector which is a 3 element array of 8-byte per element. The total comes out to \(O(168g_ag_r)\) bytes of data. We then add information describing control points: \(O(32|P_a||P_r|)\), where 32 is an 8 bytes describing weight and 24 bytes describing position. The total memory storage and transmission requirements for the first approach is \(O(168g_ag_r + 32|P_a||P_r|)\) or \(O(m^2)\), where \(m = \max(g_a, g_r)\).

For the second approach, the upper time bound on the amortized computational costs are \(O(ng_ag_r)\) where \(n\) is the total number of distinct changes done to the surface.

After running tests on a 7x7 NURBS surface with \(g_a = g_r = 50\) we calculated that using the first approach 188KB of data were transferred every time a change was made, while using a second approach it took only under 0.5 seconds to recompute the whole surface and only 100 bytes of data to be sent over to the MUGServer.

After considering the above statistics we have decided to use the second approach: instead of storing complete objects, we store the forward and backward changes that are applied to the objects. These changes are propagated through the network when we work in a multi-user mode. When the NURBS library was built, the same approach was used: whenever a control point is moved only information describing this control point is sent, such as its object-id, its previous position and its new position are stored, while the data that describes the NURBS mesh is calculated on each machine individually.

**WORKING WITH MUG**

**Example Modeling Session**

A modeling session consists of one or more designers working on a particular design. Figure 6 shows a portion of an example modeling session with two users manipulating a surface, showing the communication between the clients. In the first step, user one (on the left) pulls up a corner control point. The change to the particular control point is sent to JavaSpaces, then sent to the design agent, which then modifies it so the other clients are able to see the change and places it back into the JavaSpace, user 2 then retrieves the change from the JavaSpace and its surface is recomputed to reflect the change. In step two, user 2 pulls a front control point down, and the same communication follows as in step one to user 1. In the third, user 1 pulls up the back corner control point as user 2 changes the weight of the control point he/she just moved. In this case both changes are sent to the space and then to the design agent, once the design agent puts the changes back into the space, both users will have their surface recomputed to reflect the change the other user made. In step four, both try to pull down the center control point. When this happens, both changes are sent to the JavaSpaces and then to the design agent and which ever change is recieved last, takes effect. A session continues on like this, with both users making changes to the design, add more surfaces and curves, until both are satisfied with the result.

From this short example we can see how each user can affect the shared design, as well as how the system behaves in certain cases when contention for a control point is encountered. There are few situations which are not represented in the example session, such as when clients are concurrently manipulating different control points but affecting overlapping portions of the curve or surface. In this and other similar cases, each client’s change will be applied as in step three of the example modeling session, with the overall effect being that each client’s copy of the NURBS form is recomputed each time a change is retrieved.

**Usability Issues**

Usability issues were taken into consideration in the design and development of MUG. A formative evaluation study (Czajkowski et al., 2001) was performed on the previous version of the application and the results were taken into consideration. Currently MUG has only been tested on a local network with 2-3 simultaneous users working on a single design. Manipulations of the NURBS objects occurred almost simultaneously for each of the users as they were working on different parts of the design. Network delay, as well as computational costs for computing object shapes, were negligible. Controlled experiments on wide-area and congested networks are subjects for future study.

**System Requirements**

The present version of MUG runs on Pentium II/III-based machines with Microsoft Windows 98/NT 4.0/2000 and Sun Ultra workstations with Solaris 2.6/7 that are equipped with Java Runtime Environment version 1.2 and the VRML97, Java3D, Jini/JavaSpaces, Java Help, and XML 1.0 extensions.

**DISCUSSION AND CONCLUSIONS**

This paper presented MUG, our environment for collaborative conceptual design. Unlike most other existing conceptual design environments, MUG is multi-user and enables teams to rapidly create 3D assembly layouts and specify the structural semantics of their models. This paper has emphasized one of MUG’s novel features: the ability for multiple users to simultaneously manipulate a single mathematical surface object.

In the current version of MUG, the NURBS drawing and manipulation abilities are limited. We are currently working on adding a more dynamic curve and surface creation abilities, as well as facilities to group, move and add control points in real-time. Our near-term goals for MUG are to enhance its representational abilities, incorporate a set of shape modeling primitives, and test the scalability of our system across small workgroups.
We believe that MUG represents a novel approach to multi-user conceptual design. Java technologies have enabled us to create light-weight shared 3D worlds capable of operating on real design data. As network connectivity continues to radically reshape the design process, environments such as these will enable new design techniques based on intimately shared design spaces. We believe that our approach illustrates one channel in which collaborative engineering systems have begun to merge with computer-supported cooperative work environments.

**Acknowledgements.** This work was supported in part by National Science Foundation (NSF) Knowledge and Distributed Intelligence in the Information Age (KDI) Initiative Grant CISE/1S-9873005; CAREER Award CISE/1S-973545 and Grant ENG/DMI-9713718.

All opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the other supporting government and corporate organizations.
REFERENCES


