# Imposing Geometric Constraints on Virtual Objects within an Immersive Modeler

**Norikazu Hiraki, Kiyoshi Kiyokawa, Haruo Takemura and Naokazu Yokoya**

Graduate School of Information Science,
Nara Institute of Science and Technology (NAIST)
8916-5 Takayama, Ikoma, Nara, 630-01, Japan
**{norika-h, kiyosi-k, takemura, yokoya}@is.aist-nara.ac.jp**

**Abstract**
This paper describes an intuitive and efficient way to design not only 3-D shapes but also their geometric constraints within an immersive modeler. The fundamental concept of the proposed method is based on our immersive modeler VLEGO II, which allows multiple users to create 3-D virtual objects only by assembling simple 3-D geometric shapes (*shape primitives*) like with real toy blocks. In our system, geometric constraints are presented as primitives (*constraining primitives*) like normal shape primitives, and the geometric constraints are imposed on virtual objects simply by assembling shape and constraining primitives. The system also provides flexible two-handed interaction to support intuitive and efficient manipulations.

**Key words:** virtual reality, immersive modeler, geometric constraints, two-handed interaction, manipulation aid.

## 1 Introduction

Most real objects have geometric constraints on their movable range in relation to other adjoining objects. For example, a revolving door turns round its pillar, a drawer moves along its cabinet. Therefore, it is desirable for a modeling tool to have the ability to design not only the shape of 3-D objects, but also their geometric constraints. To support this ability with high level flexibility, a few systems employ special script languages in which a designer describes geometric constraints[1, 2]. These systems enable a designer to specify variable 3-D objects including geometric constraints, interaction and animation. However, a designer must learn the languages to describe geometric constraints of all 3-D objects, and those scripts cannot be modified interactively. Therefore, these systems don't provide an intuitive interface and require training time to use.

On the other hand, to provide an intuitive and efficient way of designing 3-D objects, virtual reality technology has attracted many researchers. Designing 3-D objects within a virtual environment rather than with 2-D projection has a number of advantages. Real-time head-tracked stereoscopic view improves the understanding of shape and spatial relationships. Spatial direct manipulation improves the accessibility to 3-D objects in a quick and intuitive way. For the last decade, a number of immersive modelers have been developed[3, 4, 5, 6]. However, most of them are only able to construct the shape of virtual objects, not to describe their behavior or geometric constraints.

We have developed a new method for imposing geometric constraints on virtual objects within an immersive modeler. This method provides a number of geometric constraints as visible primitives. Each constraining primitive has its shape, which visually represents its attribute, like other virtual objects. Hence, the attributes of geometric constraints are comprehensible. Moreover, it is easy and efficient to impose geometric constraints on virtual objects, because a user has only to combine constraining primitives with normal virtual objects using simple two-handed spatial manipulations. Our system has the following two characteristics:

1) The system has the ability of designing both 3-D objects and its geometric constraints,
2) The system provides flexible two-handed manipulation.

Owing to these characteristics, a user is able to design both 3-D objects and their geometric constraints in an intuitive and efficient manner.

This paper is organized as follows. First, previous related works for 3-D graphics with constraints are briefly reviewed. Next, the advantages of immersive modeling is described, and VLEGO II is also summarized. Then, our approach for imposing geometric constraints on virtual objects within an immersive environment is described. Finally, the implementation of our developing system is explained.

## 2 Related Works

A number of systems support designing both 3-D shapes and their behaviors. For example, TBAG[1] is a paradigm and toolkit for rapid prototyping of interactive, animated 3D graphics. TBAG is based on two broadly applied design principles: graphical abstract data types and explicit functions of time. Since TBAG is provided as C++ libraries, a user is able to describe variable 3-D objects including interaction, animation and constraints in C++. Obliq-3D[2] is a high-level, fast-turnaround system for building 3D animations. Obliq-3D consists of an interpreted language that embedded into a 3D animation library. These systems provide a large number of functions and procedures in special languages for description of a variety of properties of 3-D objects. Hence, a skilled programmer is able to describe 3-D objects with considerably complex geometric constraints and animation. However, users must describe all of properties and shapes of 3-D objects in a special language, and these description cannot be modified interactively. Therefore, these systems don't provide intuitive interfaces and require training time to use.

A large number of conventional computer aided design (CAD) tools which use 2-D input and output devices support both 3-D shapes and constraints. However, 2-D input/output devices don't provide intuitive perception and operations of 3-D objects.

On the other hand, as an intuitive and efficient way for designing 3-D objects, immersive modeling has attracted much attention because of its advantages explained in the next section. For the last decade, a number of immersive modeling systems have been developed; for example, VLEGO [4], the Conceptual Design Space (CDS) [5], and the Chapel Hill Immersive Modeling Program (CHIMP) [6]. However, most of these systems support only constructing 3-D shapes, not imposing geometric constraints on the 3-D shapes.

### 2.1 Immersive Modeling

Immersive modeling is a technique to design 3-D objects in an immersive virtual environment, while traditional CAD tools use 2-D input and output devices. Immersive modeling has two key advantages over traditional CAD tools: the stereoscopic view of a virtual world with a head-tracking facility and the spatial direct manipulations of 3-D virtual objects. Both of them help to avoid ambiguous perception and complicated 2-D operations.

The stereoscopic view of a virtual world is generally provided by a HMD (Head Mounted Display) or a pair of liquid crystal shuttered glasses. The stereoscopic view helps users to understand shapes of 3-D objects and spatial relationships among the objects in a virtual world. Moreover, real-time head-tracking facility with a 3-D tracker provides motion parallax, which makes it easier to perceive 3-D objects.

Spatial direct manipulation is usually implemented by 6DOF (3-D position and orientation) input devices (3-D mice, gloves with a 3-D tracker, etc.) held in hands. Since a user is able to interact directly with 3-D objects in a virtual world based on

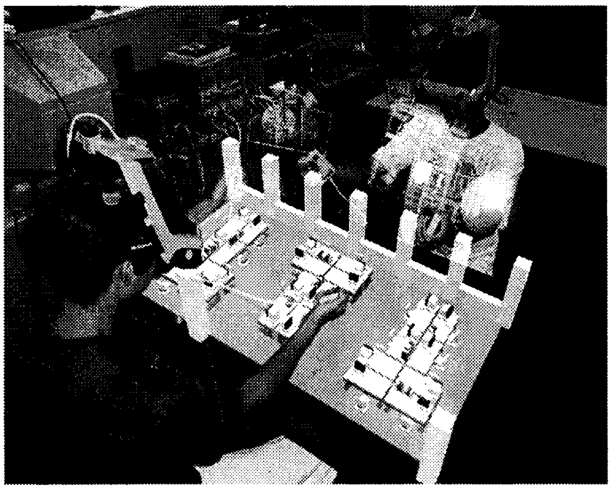

Figure 1: VLEGO II workspace.

proprioception, spatial direct manipulation provides the user with intuitive and quick means to access the 3-D objects.

In addition to the advantages above, a user can easily confirm the results of modification of the 3-D objects, because immersive modeling enables the user to design 3-D objects entirely within a virtual world. However, manipulation in an immersive environment tends to be awkward and fatiguing owing to lack of force-feedback, computational delay and so on. This disadvantage should be eliminated and certain software based manipulation aid is serviceable.

### 2.2 VLEGO II

We have been developing an immersive modeler VLEGO II, a shared virtual environment for collaborative two-handed 3-D modeling (Figure 1). VLEGO II is implemented on one or two SGI graphics workstation(s). The virtual workspace is displayed stereoscopically to a user through a HMD with a head-tracking facility. A user holds a pair of 3-D input devices in his or her hands. Each device has a 6DOF magnetic tracker 3SPACE (Polhemus) and four feather touch switches on it. These devices are used to manipulate virtual objects via two 3-D cursors. In VLEGO II, all operations on 3-D objects are performed by using the 3-D cursors.

In VLEGO II, all virtual objects consist of one or more simple 3-D geometric shapes (spheres, boxes, cones, cylinders, etc.) called *shape primitives.* Primitives are stored in a primitive box in advance and they are generated unlimitedly from the box simply by picking the primitives in the box. Since a user can construct 3-D objects simply by assembling the primitives like real toy blocks, he or she need not learn complex operations.

VLEGO II has the following three characteristics:

(1) A number of flexible two-handed manipulations such as assembly, decomposition, coloring

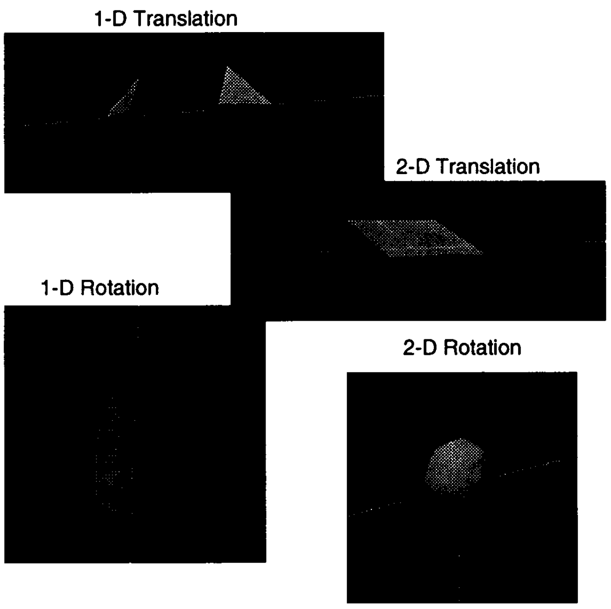

Figure 2: Constraining primitives.

and scaling objects.

(2) Discrete constraints, which restrict the position and orientation of 3-D objects discretely to make it easy to arrange.

(3) Collision avoidance, which detects collisions among 3-D objects and adjusts their locations.

VLEGO II offers natural and quick ways for multiple users to create 3-D virtual objects cooperatively in a shared virtual workspace. Our new system fundamentally employs the designing concept of VLEGO II.

## 3 Constraining Primitives

A variety of types of constraints can be considered for constructing 3-D objects. For example, geometric constraints, time constraints, kinetic constraints, etc. In this paper, we focus on geometric constraints, which most real objects have.

In our developing system, geometric constraints are represented as primitives (*constraining primitives*) like normal shape primitives. Each constraining primitive has its own 3-D shape, which visually represents its attributes, e.g., the constraint type, the principal axes, etc. Therefore, a user is able to easily understand the behavior of the constraints.

We have begun with simple translation and rotation constraints. Currently, the system has the following four types of constraining primitives illustrated in Figure 2.

- **1-D translation:** This primitive has two gray pyramids and a red line. The red line represents an axis. An object constrained by this primitive can move parallel to the axis.
- **2-D translation:** This primitive has a gray plate and two red lines. These lines define a plane which constrained object can be translated parallel to.
- **1-D rotation:** This primitive has a gray cylinder and a red line. The red line represents an axis, and an object constrained by this primitive rotates around the axis.
- **2-D rotation:** This primitive has a gray sphere and two red lines. The longer and shorter lines correspond to axes of an azimuth and an elevation respectively. An object constrained by this primitive rotates at the center of the sphere.

Users can make all constraining primitives invisible, since shapes of these primitives are originally nothing to do with the intended design.

The constraining primitives can be treated like shape primitives in terms of 3-D manipulation. Imposing geometric constraints on virtual objects is accomplished by simply assembling shape and constraining primitives. Therefore, a user need not learn complex manipulations. Furthermore, since a user works entirely within the virtual world, the result of modified attributes of geometric constraints and imposed geometric constraints on 3-D objects can be confirmed in real time.

## 4 Implementation

This section describes the implementation of the system that is based on the concept presented in the previous section. In the following, hardware components, typical manipulations and a number of examples constructed in the system are explained.

### 4.1 Hardware

Figure 3 illustrates the hardware components of the system. The system is implemented on a graph-

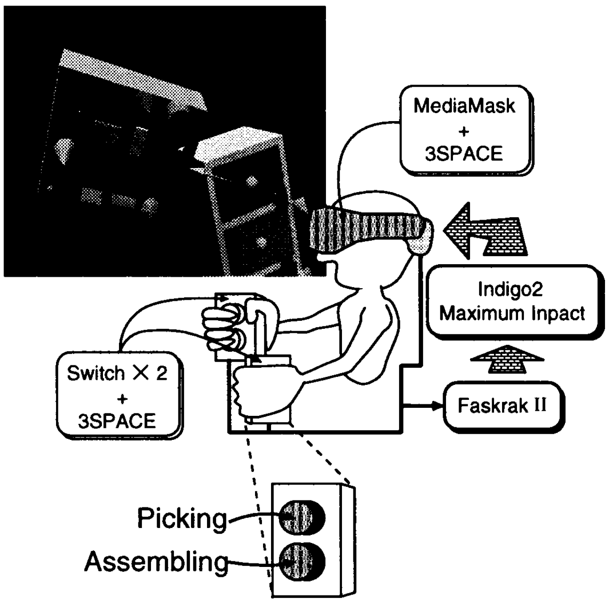

Figure 3: Hardware components.

ics workstation Indigo2 Maximum Impact (SGI). A user is able to view a virtual world stereoscopically through a HMD MediaMask (Olympus) with a real-time head-tracking facility using a 3-D tracker 3SPACE Fastrak II (Polhemus). A pair of hand-held 6DOF input devices using 3-D trackers allows a user to manipulate two 3-D cursors to operate 3-D objects, and each of the devices has two feather touch switches. One switch is used to pick 3-D objects, and the other is mainly used to assemble 3-D objects. We will refer to the former as *picking button*, and the latter as *assembling button*.

### 4.2 Basic Manipulations

We now describe typical manipulations to interact with 3-D objects using two 3-D cursors in the system. As described earlier, the system is fundamentally based on VLEGO II, so the manipulations in the system are similar to those manipulations in VLEGO II.

- **Selection:** When a tip of a 3-D cursor is positioned in a primitive, it is selected. To make the selection clear, the stabbed primitive is highlighted and a bounding box of the object appears.
- **Picking:** When an object is selected and the picking button is pushed, the object is picked and held. While holding the object, a user is able to translate and rotate it freely. The picked object is released by releasing the picking button.

The system supports natural two-handed manipulations. For example,

- **Scaling:** When an object is picked by two 3-D cursors at the same time, the object can be scaled.

To help a user to arrange 3-D objects, the system employs discrete placement constraints. Manipulating virtual objects tends to be awkward because of lack of force-feedback, computational delay, restricted spatial resolution of input/output devices. So, in order to make the interface feasible without bulky force-feedback devices, the system imposes simple and comprehensible constraints on 3-D objects' movement if no other constraints are imposed. In the system, 3-D objects can be located at discrete positions at intervals of 1cm, and its orientation is restricted at every 90 degrees.

### 4.3 Design Process

The process for constructing constrained 3-D objects in the system is generally as follows.

1. Pick up proper primitives from a primitive box.
2. Assemble these primitives into desirable 3-D parts.
3. Impose geometric constraints on each part.

The following describes each of these steps.

#### 4.3.1 Picking Up Primitives

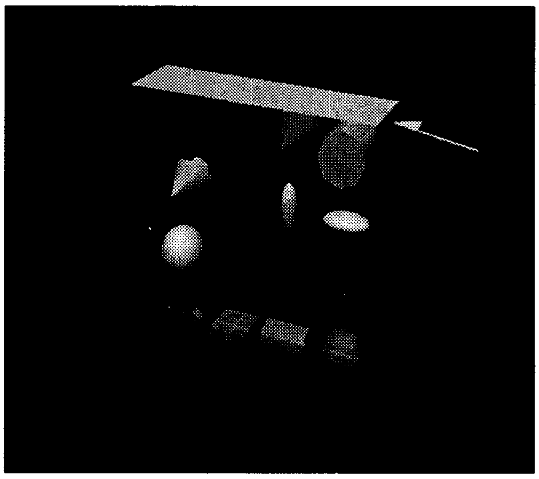

Figure 4: A primitive box.

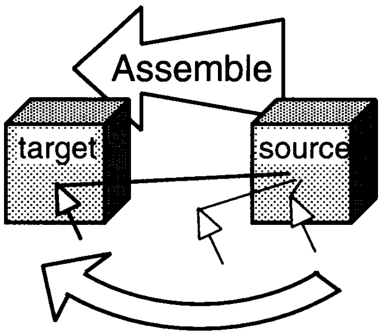

Figure 5: One-handed Assembling.

In the system, each 3-D object consists of one or more primitives. Several kinds of primitives including both shape primitives and constraining primitives are contained in a primitive box floating in the virtual workspace (Figure 4). These primitives are generated unlimitedly by simply picking the primitives in the box. Additionally, when a picked 3-D object is released in the primitive box, the object is registered. Once a 3-D object is registered, the registered object can be copied unlimitedly just as the primitives registered in advance.

#### 4.3.2 Assembling 3-D Shapes

In the system, two types of manipulations for assembling 3-D objects are implemented. One is a one-handed manipulation, and the other is a cooperative two-handed manipulation.

Figure 5 illustrates the one-handed assembling manipulation. First, select a 3-D object (we will call this object the *source object* for explanation), and push the assembling button. After these actions, a line appears from the tip of 3-D cursor to the center of the source object. Next, drag the 3-D cursor into a desirable 3-D object (we will call this object the *target object*), and release the assembling button. Then, the source object is sticked onto the target object, and the line disappears.

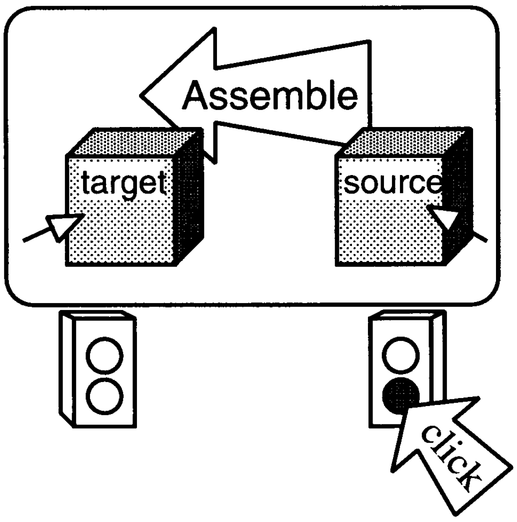

Figure 6: Two-handed Assembling.

The other assembling manipulation is illustrated in Figure 6. This manipulation causes the same result as the one-handed assembling manipulation. First, select or pick the source object and the target object with each hand. Next, click the assembling button on the input device which selects (picks) the source object. Then, the source object is sticked onto the target object.

As a result of these assembling manipulations, the source object and the target object are grouped.

### 4.3.3 Imposing Geometric Constraints

To impose geometric constraints on 3-D objects, a user has only to assemble a constraining primitive and a 3-D object using the same assembling manipulation explained above. In the assembling manipulation, when the target object is a constraining primitive, the source object is constrained by the target object according to the attributes of the constraining primitive. Therefore, a user need not learn special manipulations for imposing geometric constrains, and he or she is able to impose geometric constraints on 3-D object easily.

Figure 7 shows an example of imposing geometric constraints. This example illustrates a construction of a revolving door that consists of two shape primitives (a door and a wall) and a 1-D rotation constraining primitive. First, when the door is sticked onto the constraining primitive (Figure 7-a), the door is constrained by the constraining primitive. That is, the door turns around the axis represented by the constraining primitive (Figure 7-b). When the constraining primitive is sticked onto the wall (Figure 7-c), the constraining primitive and the wall are grouped and move together (Figure 7-d). As a result of these two actions, the wall substantially constrains the door through the constraining primitive (Figure 7-e).

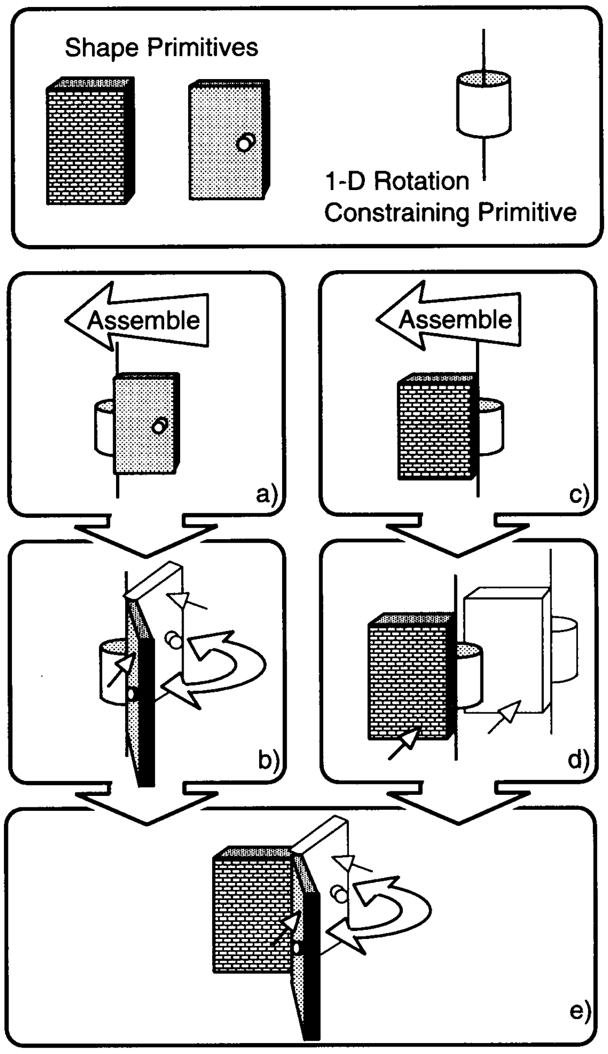

Figure 7: An example of imposing geometric constraints.

## 4.4 Examples

A number of examples constructed in the system are presented below.

- **Cabinet:** Figure 8 illustrates a cabinet, which consists of a number of boards and a 1-D translation constraining primitive. Each drawer is constrained its movement by the constraining primitive.
- **ConeTree:** Figure 9 shows a simple ConeTree that is a method of 3-D visualization of hierarchical structure [7]. This object is composed of two 1-D rotation constraining primitives, two cones and twelve spheres. Our system is also useful for rapid prototyping of 3-D user interface components.
- **Puppets:** The example shown in Figure 10 contains two puppets. Each puppet consists of fifteen spheres and fourteen 2-D rotation constrain-

ing primitives, which are imposed on each joint. In this example, the constraining primitives are invisible.

## 5 Conclusions and Future Works

In this paper, we have presented a simple and efficient way to impose geometric constraints on virtual objects within an immersive modeler. In our immersive modeling system, geometric constraints are presented as constraining primitives which have shapes representing their attributes. The geometric constraints are imposed on virtual objects by simply assembling shape primitives and constraining primitives. This approach makes it intuitive and easy to impose geometric constraints on virtual objects. As a result, a user can easily design both 3-D shapes and their geometric constraints within the virtual workspace.

Future works include the following two themes. First, propagation of constraints and dissolution of inconsistency should be investigated. Propagation of constraints allows a user to create more complicated 3-D objects, but may raise the inconsistency between the constraints. As a matter of course, this inconsistency must be detected and eliminated. Second, we should consider other types of constraints; e.g., time constraints that help to construct a dynamic virtual world including animations, kinetic constraints like a spring model, etc.

**References**

[1] C. Elliott, G. Schechter, R. Yeung, and S. Abi-Ezzi: "TBAG: A High Level Framework for Interactive, Animated 3D Graphics Applications," *Proc. ACM SIGGRAPH'94*, pp.421-434, 1994.

[2] M. A. Najork and M. H. Brown: "Obliq-3D: A High-Level, Fast-Turnaround 3D Animation System," *IEEE Transactions on Visualization and Computer Graphics*, Vol.1, No.2, pp.175-193, 1995.

[3] J. Butterworth, A. Davidson, S. Hench and T. M. Olano: "3DM: A Three Dimensional Modeler Using a Head-Mounted Display," *Proc. of ACM Simpo. on Interactive 3D Graphics*, pp.135-139, 1992.

[4] K. Kiyokawa, T. Haruo, Y. Katayama, H. Iwasa and N. Yokoya.: "VLEGO: A Simple Two-Handed Modeling Environment Based on Toy Blocks," *Proc. of ACM Simpo. on Virtual Reality Software and Technology (VRST'96)*, pp.27-34, 1996.

[5] D. A. Bowman and L. F. Hodges: "User Interface Constraints for Immersive Virtual Environment Applications," Graphics, Visualization and Usability Center Technical Report, GIT-GVU-95-26, 1995.

[6] M. R. Mine: "Working in a Virtual World: Interaction Techniques Used in the Chapel Hill Immersive Modeling Program," UNC Chapel Hill Computer Science Technical Report, TR96-029, 1996.

[7] G. G. Robertson, J. D. Mackinlay, and S. K. Card: "ConeTree: Animated 3D Visualizations of Hierarchical Information," *Proc. of SIGCHI'91: the Conference on Human Factors in Computing Systems*, pp.189-194, 1991.

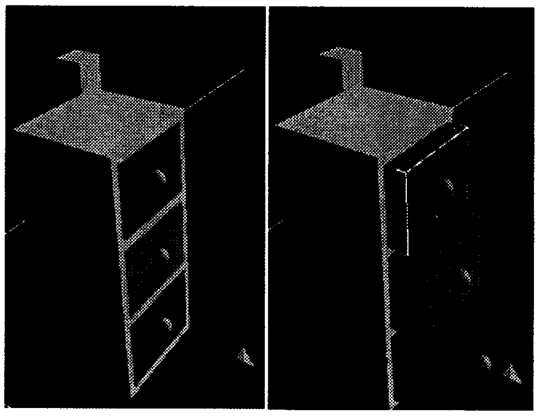
Figure 8: A cabinet.

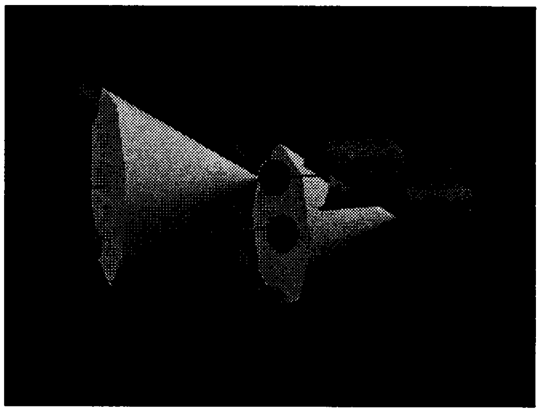
Figure 9: A ConeTree.

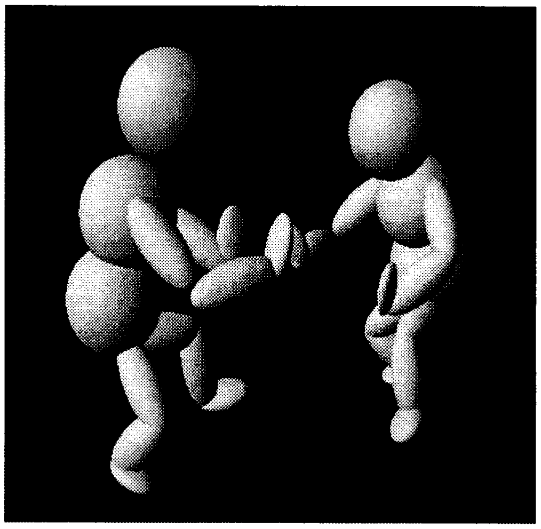
Figure 10: Puppets.