# Interaction

## Interfaces, Algorithms, and Applications

**SIGGRAPH 2009, Course #6**
**Friday, August 7 2009**

**8:30 AM - 12:15 PM, Auditorium A**

**http://www.gmrv.es/sig09_interaction/**


Miguel A. Otaduy
URJC Madrid
miguel.otaduy@urjc.es

Takeo Igarashi
The University of Tokyo, JST/ERATO
takeo@acm.org

Joseph J. LaViola Jr.
University of Central Florida
jjl@cs.ucfl.edu

# Contents

# 1 Course Structure

The field of virtual reality has recently seen the advent of novel commodity 3D user interfaces that have led to not only a revolution in video game interaction, but also new possibilities for other virtual reality applications. This course provides background on the interfaces and algorithms involved in 3D interaction, and also gives an outlook into the research in the field.

The course will cover interaction algorithms, their application to animation, interactive manipulation, video games, and sketch-based interfaces. The course will also pay special attention to interfaces and to tactile interaction techniques. It is intended to programmers and researchers in computer graphics working on interactive applications, such as modeling, virtual reality, videogames, etc., interested in particular on intuitive interaction techniques.

The course will be divided into three large areas:

- User interfaces and interaction techniques.
- Applications involving 3D interaction and sketching.
- Tactile interaction with virtual environments.

The first part of the course will cover input and output technologies that make 3D interaction possible. The field is well covered in research nowadays, but the progress in interaction technology makes designing effective 3D interfaces an evolving skill. The first part will also pick video games as a particular application case, and describe the use of 3D user interfaces in that context.

The second part will cover the use of sketch-based interfaces in a large number of applications in computer graphics. It will demonstrate how 3D interaction techniques have been successfully applied to, e.g., 3D shape modeling, animation, or special purpose editors.

The third and final part of the course will reflect the importance of tactile interaction on user interfaces. They provide important features such as situation awareness and the ability to perform detailed manipulations. Effective tactile interaction requires basic knowledge of psychophysics aspects, control theory, and physically-based simulation, which are overviewed in the course.

# 2 3D User Interfaces and Video Games

## 2.1 Introduction

3D user interface (3D UI) research has been going on for more than 20 years, with limited success in being a part of people's everyday lives. Although 3D UIs have been useful in areas such as simulation and training, scientific visualization, medicine, and psychological rehabilitation, they have not significantly affected mainstream society. Compared to other technologies such as the mouse and WIMP (windows, icons, menus, point and click) interfaces, 3D UIs have had little impact. However, we're seeing a change in the utility of virtual reality (VR) and 3D UIs, with concepts from these fields emerging on a large, mainstream scale. The video game industry, as it has done for real-time 3D computer graphics for the last 15 years, is bringing spatial 3D interaction into people's living rooms through rather simple, yet effective technologies such as computer vision and motion sensing. The popularity of devices such as the Sony EyeToy and Nintendo Wii have shown that 3D UIs are here to stay and will continue to evolve to enhance the power and playability of video games, producing rich, immersive, and engaging user experiences [LaViola Jr. 2008].

### 2.1.1 What is a 3D UI?

A 3D user interface is a UI that involves 3D interaction, human computer interaction where the user's tasks are carried out in a 3D spatial context with 3D input devices or 2D input devices with direct mappings to 3D. Typically, 3D interaction techniques, methods (hardware and software), allowing a user to accomplish a spatial task make up the components of a 3D UI [Bowman et al. 2004]. Figure 1 shows example 3D UIs.

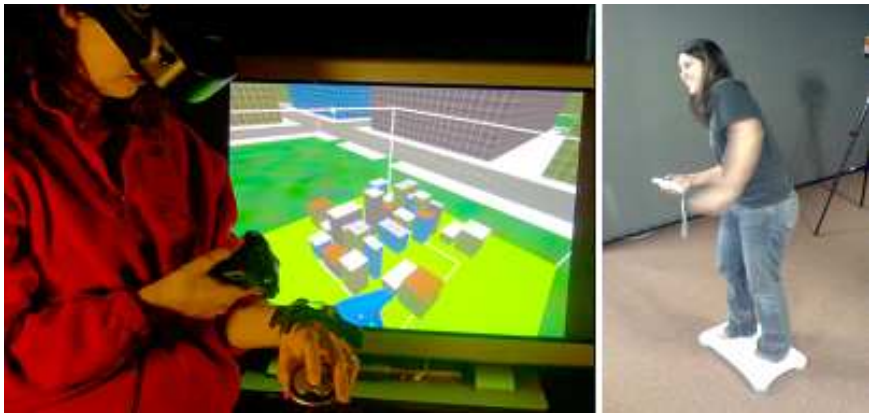

**Figure 1:** *Two examples of a 3D UI. The image on the left shows a user interacting with a world-in-miniature and the image on the right shows a user navigating using body-based controls.*

### 2.1.2 3D UIs in Video Games

As discussed in Section 2.1.1, 3D UIs involve input devices and interaction techniques for effectively controlling highly dynamic 3D computer-generated content, and there's no exception when it comes to video games. A 3D video game world can use one of three basic approaches to interaction. The first maps 2D input and button devices, such as the keyboard and mouse, joysticks, and game controllers to game elements in the 3D world. This is the traditional approach; it's how people have been interacting with 3D (and 2D) video games since their inception many years ago. The second approach simulates the real world using replicas of existing devices or physical props. Common examples include steering wheels, light guns, and musical instruments (for example, the guitar in Guitar Hero). These devices don't necessarily provide 3D interaction in the game but do provide 3D input devices that enable more realistic gameplay. The third approach is true spatial 3D tracking of the user's motion and gestures,

where users interact in and control elements of the 3D gaming world with their bodies. This control can come through vision-based devices such as the Sony EyeToy and motion-sensing devices such as Nintendo Wii Remotes (Wiimotes).

### 2.1.3 Lecture Outline

In the remainder of this lecture, we will first examine 3D user interfaces in general terms by looking at common interaction techniques that have been developed. In the second part of the lecture, we will discuss 3D UIs in the video game domain, with a focus on interfaces and video game prototypes developed using Wiimotes.

## 2.2 3D Spatial Interaction Techniques

There are essentially four basic 3D interaction tasks that are found in most complex 3D applications. Obviously, there are other tasks which are specific to an application domain, but these basic building blocks can often be combined to let users perform more complex tasks. These tasks include navigation, selection, manipulation, and system control. **Navigation** is the most common VE task, and is consists of two components. *Travel* is the motor component of navigation, and just refers to physical movement from place to place. *Wayfinding* is the cognitive or decision-making component of navigation, and it asks the questions, ”where am I?”, ”where do I want to go?”, ”how do I get there?”, and so on. **Selection** is simply the specification of an object or a set of objects for some purpose. **Manipulation** refers to the specification of object properties (most often position and orientation, but also other attributes). Selection and manipulation are often used together, but selection may be a stand-alone task. For example, the user may select an object in order to apply a command such as ”delete” to that object. **System control** is the task of changing the system state or the mode of interaction. This is usually done with some type of command to the system (either explicit or implicit). Examples in 2D systems include menus and command-line interfaces. It is often the case that a system control technique is composed of the other three tasks (e.g. a menu command involves selection), but it’s also useful to consider it separately since special techniques have been developed for it and it is quite common.

In this section, we will review several common techniques used to perform these basic tasks. Note that it is beyond the scope of this lecture to go into great detail on these techniques or on the concepts of 3D UIs in general. The reader should examine ”3D User Interfaces: Theory and Practice” for a rigorous treatment of the subject [Bowman et al. 2004]. Also note for this section, we assume (for the techniques where it makes a difference) that ’y’ is the vertical axis in the world coordinate system.

### 2.2.1 Navigation

The motor component of navigation is known as travel (e.g., viewpoint movement). There are several issues to consider when dealing with travel in 3D UIs. One such issue is the control of velocity and/or acceleration. There are many methods for doing this, including gesture, speech controls, sliders, etc. Another issue is that of world rotation. In systems that are only partially spatially surrounding (e.g. a 4-walled CAVE, or a single screen), the user must be able to rotate the world or his view of the world in order to navigate. In fully surrounding systems (e.g. with an HMD or 6-sided CAVE) this is not necessary since the visuals completely surround the user. Next, one must consider whether motion should be constrained in any way, for example by maintaining a constant height or by following the terrain. Finally, at the lowest-level, the conditions of input must be considered - that is, when and how does motion begin and end (click to start/stop, press to start, release to stop, stop automatically at target location, etc.)? Four of the more common 3D travel techniques are gaze-directed steering, pointing, map-based travel, and ”grabbing the air”.

**Gaze-Directed Steering** Gaze-directed steering is probably the most common 3D travel technique, although the term ”gaze” is really misleading. Usually no eye tracking is being performed, so the direction of gaze is inferred

from the head tracker orientation. This is a simple technique, both to implement and to use, but it is somewhat limited in that you cannot look around while moving [Mine 1995].

To implement gaze-directed steering, typically a callback function is set up that executes before each frame is rendered. Within this callback, first obtain the head tracker information (usually in the form of a 4x4 matrix). This matrix gives you a transformation between the base tracker coordinate system and the head tracker coordinate system. By also considering the transformation between the world coordinate system and the base tracker coordinates (if any), you can get the total composite transformation. Now, consider the vector $(0, 0, -1)$ in head tracker space (the negative z-axis, which usually points out the front of the tracker). This vector, expressed in world coordinates, is the direction you want to move. Normalize this vector, multiply it by the speed, and then translate the viewpoint by this amount in world coordinates. Note: current "velocity" is in units/frame. If you want true velocity (units/second), you must keep track of the time between frames and then translate the viewpoint by an amount proportional to that time.

**Pointing** Pointing is also a steering technique (where the user continuously specifies the direction of motion). In this case, the hand's orientation is used to determine direction. This technique is somewhat harder to learn for some users, but is more flexible than gaze-directed steering [Bowman et al. 1997]. Pointing is implemented in exactly the same way as gaze-directed steering, except a hand tracker is used instead of the head tracker.

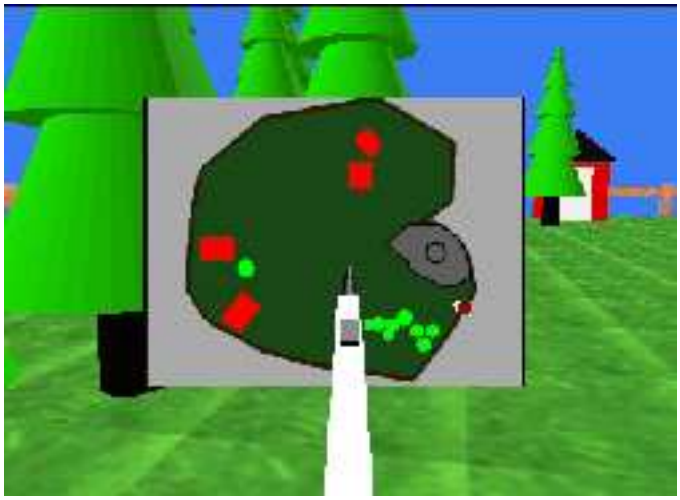

**Figure 2:** *Dragging a user icon to move to a new location in the world.*

**Map-based Travel** The map-based travel technique is a target-based technique. The user is represented as an icon on a 2D map of the environment. To travel, the user drags this icon to a new position on the map (see Figure 2). When the icon is dropped, the system smoothly animates the user from the current location to the new location indicated by the icon [Bowman et al. 1998].

To implement this technique, two things must be known about the way the map relates to the world. First, we need to know the scale factor, the ratio between the map and the virtual world. Second, we need to know which point on the map represents the origin of the world coordinate system. We assume here that the map model is originally aligned with the world (i.e. the x direction on the map, in its local coordinate system, represents the x direction in the world coordinate system). When the user presses the button and is intersecting the user icon on the map, then the icon needs to be moved with the stylus each frame. One cannot simply attach the icon to the stylus, because we want the icon to remain on the map even if the stylus does not. To do this, we first find the position of the stylus in the map coordinate system. This may require a transformation between coordinate systems, since the stylus is not a child of the map. The x and z coordinates of the stylus position are the point to which the icon should be moved.

We do not cover here what happens if the stylus is dragged off the map, but the user icon should "stick" to the side of the map until the stylus is moved back inside the map boundaries, since we don't want the user to move outside the world.

When the button is released, we need to calculate the desired position of the viewpoint in the world. This position is calculated using a transformation from the map coordinate system to the world coordinate system, which is detailed here. First, find the offset in the map coordinate system from the point corresponding to the world origin. Then, divide by the map scale (if the map is 1/100 the size of the world, this corresponds to multiplying by 100). This gives us the x and z coordinates of the desired viewpoint position. Since the map is 2D, we can't get a y coordinate from it. Therefore, the technique should have some way of calculating the desired height at the new viewpoint. In the simplest case, this might be constant. In other cases, it might be based on the terrain height at that location or some other factors.

Once we know the desired viewpoint, we have to set up the animation of the viewpoint. The move vector $\mathbf{m}$ represents the amount of translation to do each frame (we are assuming a linear path). To find $\mathbf{m}$, we subtract the desired position from the current position (the total movement required), divide this by the distance between the two points (calculated using the distance formula), and multiplied by the desired velocity, so that $\mathbf{m}$ gives us the amount to move in each dimension each frame. The only remaining calculation is the number of frames this movement will take: distance/velocity frames. Note that again velocity is measured here in units/frame, not units/second, for simplicity.

**Grabbing the Air** The grabbing the air technique uses the metaphor of literally grabbing the world around you (usually empty space), and pulling yourself through it using hand gestures [Mapes and Moshell 1995]. This is similar to pulling yourself along a rope, except that the rope exists everywhere, and can take you in any direction.

To implement the one-handed version of this technique (the two-handed version can get complex if rotation and world scaling is also supported), when the initial button press is detected, we simply obtain the position of the hand in the world coordinate system. Then, every frame until the button is released, get a new hand position, subtract it from the old one, and move the objects in the world by this amount. Alternately, you can leave the world fixed, and translate the viewpoint by the opposite vector. Before exiting the callback, be sure to update the old hand position for use on the next frame. Note it is tempting to implement this technique simply by attaching the world to the hand, but this will have the undesirable effect of also rotating the world when the hand rotates, which can be quite disorienting. You can also do simple constrained motion simply by ignoring one or more of the components of the hand position (e.g. only consider x and z to move at a constant height).

### 2.2.2 Selection

3D selection is the process of accessing one or more objects in a 3D virtual world. Note that selection and manipulation are intimately related, and that several of the techniques described here can also be used for manipulation. There are several common issues for the implementation of selection techniques. One of the most basic is how to indicate that the selection event should take place (e.g. you are touching the desired object, now you want to pick it up). This is usually done via a button press, gesture, or voice command, but it might also be done automatically if the system can infer the users intent. One also has to have efficient algorithms for object intersections for many of these techniques. Well discuss a couple of possibilities. The feedback given to the user regarding which object is about to be selected is also very important. Many of the techniques require an avatar (virtual representation) for the users hand. Finally, consider keeping a list of objects that are selectable, so that a selection technique does not have to test every object in the world, increasing efficiency. Four common selection techniques include the virtual hand, ray-casting, occlusion, and arm extension.

**The Virtual Hand** The most common selection technique is the simple virtual hand, which does real-world selection via direct touching of virtual objects. In the absence of haptic feedback, this is done by intersecting the virtual hand (which is at the same location as the physical hand) with a virtual object.

Implementing this technique is simple, provided you have a good intersection/collision algorithm. Often, intersections are only performed with axis-aligned bounding boxes or bounding spheres rather than with the actual geometry of the objects.

**Ray-Casting** Another common selection technique is ray-casting. This technique uses the metaphor of a laser pointer an infinite ray extending from the virtual hand [Mine 1995]. The first object intersected along the ray is eligible for selection. This technique is efficient, based on experimental results, and only requires the user to vary 2 degrees of freedom (pitch and yaw of the wrist) rather than the 3 DOFs required by the simple virtual hand and other location-based techniques.

There are many ways to implement ray-casting. A brute-force approach would calculate the parametric equation of the ray, based on the hands position and orientation. First, as in the pointing technique for travel, find the world coordinate system equivalent of the vector $(0, 0, -1)$. This is the direction of the ray. If the hands position is represented by $(x_h, y_h, z_h)$, and the direction vector is $(x_d, y_d, z_d)$, then the parametric equations are given by

$$x(t) = x_h + x_d t \quad (1)$$
$$y(t) = y_h + y_d t \quad (2)$$
$$z(t) = z_h + z_d t. \quad (3)$$

Only intersections with $t > 0$ should be considered, since we do not want to count intersections behind the hand. It is important to determine whether the actual geometry has been intersected, so first testing the intersection with the bounding box will result in many cases being trivially rejected.

Another method might be more efficient. In this method, instead of looking at the hand orientation in the world coordinate system, we consider the selectable objects to be in the hands coordinate system, by transforming their vertices or their bounding boxes. This might seem quite inefficient, because there is only one hand, while there are many polygons in the world. However, we assume we have limited the objects by using a selectable objects list. Thus, the intersection test we will describe is much more efficient. Once we have transformed the vertices or bounding boxes, we drop the z coordinate of each vertex. This maps the 3D polygon onto a 2D plane (the xy plane in the hand coordinate system). Since the ray is $(0, 0, -1)$ in this coordinate system, we can see that in this 2D plane, the ray will intersect the polygon if and only if the point $(0, 0)$ is in the polygon. We can easily determine this with an algorithm that counts the number of times the edges of the 2D polygon cross the positive x-axis. If there are an odd number of crossings, the origin is inside, if even, the origin is outside.

**Occlusion Techniques** Occlusion techniques (also called image plane techniques) work in the plane of the image; object are selected by covering it with the virtual hand so that it is occluded from your point of view [Pierce et al. 1997]. Geometrically, this means that a ray is emanating from your eye, going through your finger, and then intersecting an object.

These techniques can be implemented in the same ways as the ray-casting technique, since it is also using a ray. If you are doing the brute-force ray intersection algorithm, you can simply define the rays direction by subtracting the finger position from the eye position.

However, if you are using the 2nd algorithm, you require an object to define the rays coordinate system. This can be done in two steps. First, create an empty object, and place it at the hand position, aligned with the world coordinate

system. Next, determine how to rotate this object/coordinate system so that it is aligned with the ray direction. The angle can be determined using the positions of the eye and hand, and some simple trigonometry. In 3D, two rotations must be done in general to align the new objects coordinate system with the ray.

**Arm-Extension** The arm-extension (e.g., Go-Go) technique is based on the simple virtual hand, but it introduces a nonlinear mapping between the physical hand and the virtual hand, so that the user's reach is greatly extended [Poupyrev et al. 1996]. The graph in Figure 3 shows the mapping between the physical hand distance from the body on the x-axis and the virtual hand distance from the body on the y-axis. There are two regions. When the physical hand is at a depth less than a threshold D, the one-to-one mapping applies. Outside D, a non-linear mapping is applied, so that the farther the user stretches, the faster the virtual hand moves away.

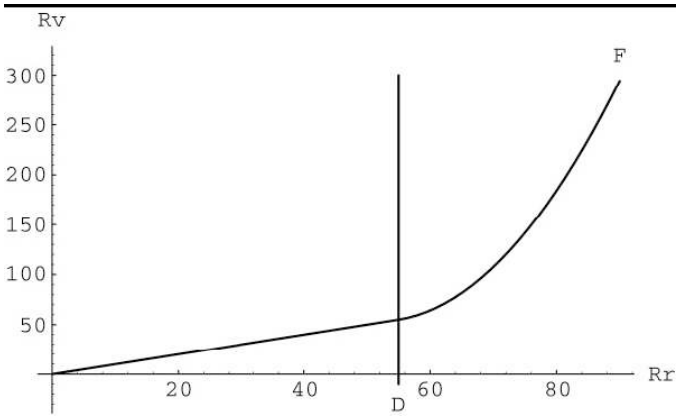


**Figure 3:** *The nonlinear mapping function used in the Go-Go selection technique.*

To implement Go-Go, we first need the concept of the position of the users body. This is needed because we stretch our hands out from the center of our body, not from our head (which is usually the position that is tracked). We can implement this using an inferred torso position, which is defined as a constant offset in the negative y direction from the head. A tracker could also be placed on the users torso.

Before rendering each frame, we get the physical hand position in the world coordinate system, and then calculate its distance from the torso object using the distance formula. The virtual hand distance can then be obtained by applying the function shown in the graph in Figure 3. $d^{2.3}$ (starting at D) is a useful function in many environments, but the exponent used depends on the size of the environment and the desired accuracy of selection at a distance. Once the distance at which to place the virtual hand is known, we need to determine its position. The most common implementation is to keep the virtual hand on the ray extending from the torso and going through the physical hand. Therefore, if we get a vector between these two points, normalize it, multiply it by the distance, then add this vector to the torso point, we obtain the position of the virtual hand. Finally, we can use the virtual hand technique for object selection.

### 2.2.3 Manipulation

As we noted earlier, manipulation is connected with selection, because an object must be selected before it can be manipulated. Thus, one important issue for any manipulation technique is how well it integrates with the chosen selection technique. Many techniques, as we have said, do both: e.g. simple virtual hand, ray-casting, and go-go. Another issue is that when an object is being manipulated, you should take care to disable the selection technique and the feedback you give the user for selection. If this is not done, then serious problems can occur if, for example, the user tries to release the currently selected object but the system also interprets this as trying to select a new object. Finally, thinking about what happens when the object is released is important. Does it remain at its last

position, possibly floating in space? Does it snap to a grid? Does it fall via gravity until it contacts something solid? The application requirements will determine this choice. Three common manipulation techniques include HOMER, Scaled-World Grab, and World-in-Miniature.

**HOMER** The Hand-Centered Object Manipulation Extending Ray-Casting (HOMER) technique uses ray-casting for selection and then moves the virtual hand to the object for hand-centered manipulation [Bowman and Hodges 1997]. The depth of the object is based on a linear mapping. The initial torso-physical hand distance is mapped onto the initial torso-object distance, so that moving the physical hand twice as far away also moves the object twice as far away. Also, moving the physical hand all the way back to the torso moves the object all the way to the users torso as well.

Like Go-Go, HOMER requires a torso position, because you want to keep the virtual hand on the ray between the users body (torso) and the physical hand. The problem here is that HOMER moves the virtual hand from the physical hand position to the object upon selection, and it is not guaranteed that the torso, physical hand, and object will all line up at this time. Therefore, we calculate where the virtual hand would be if it were on this ray initially, then calculate the offset to the position of the virtual object, and maintain this offset throughout manipulation.

When an object is selected via ray-casting, first detach the virtual hand from the hand tracker. This is due to the fact that if it remained attached but the virtual hand model is moved away from the physical hand location, a rotation of the physical hand will cause a rotation and translation of the virtual hand. Next, move the virtual hand in the world coordinate system to the position of the selected object, and attach the object to the virtual hand in the scene graph (again, without moving the object in the world coordinate system).

To implement the linear depth mapping, we need to know the initial distance between the torso and the physical hand $d_h$, and between the torso and the selected object $d_o$. The ratio $d_o/d_h$ will be the scaling factor.

For each frame, we need to set the position and orientation of the virtual hand. The selected object is attached to the virtual hand, so it will follow along. Setting the orientation is relatively easy. Simply copy the transformation matrix for the hand tracker to the virtual hand, so that their orientation matches. To set the position, we need to know the correct depth and the correct direction. The depth is found by applying the linear mapping to the current physical hand depth. The physical hand distance is simply the distance between it and the torso, and we multiply this by the scale factor $d_o/d_h$ to get the virtual hand distance. We then obtain a normalized vector between the physical hand and the torso, multiply this vector by the virtual hand distance, and add the result to the torso position to obtain the virtual hand position.

**Scaled-World Grab** The scaled-world grab technique (see Figure 4) is often used with occlusion selection. The idea is that since you are selecting the object in the image plane, you can use the ambiguity of that single image to do some magic. When the selection is made, the user is scaled up (or the world is scaled down) so that the virtual hand is actually touching the object that it is occluding. If the user does not move (and the graphics are not stereo), there is no perceptual difference between the images before and after the scaling [Mine et al. 1997]. However, when the user starts to move the object and/or his head, he realizes that he is now a giant (or that the world is tiny) and he can manipulate the object directly, just like the simple virtual hand.

To implement scaled-world grab, correct actions must be performed at the time of selection and release. Nothing special needs to be done in between, because the object is simply attached to the virtual hand, as in the simple virtual hand technique. At the time of selection, scale the user by the ratio (distance from eye to object / distance from eye to hand). This scaling needs to take place with the eye as the fixed point, so that the eye does not move, and should be uniform in all three dimensions. Finally, attach the virtual object to the virtual hand. At the time of release, the opposite actions are done in reverse. Re-attach the object to the world, and scale the user uniformly by the reciprocal of the scaling factor, again using the eye as a fixed point.

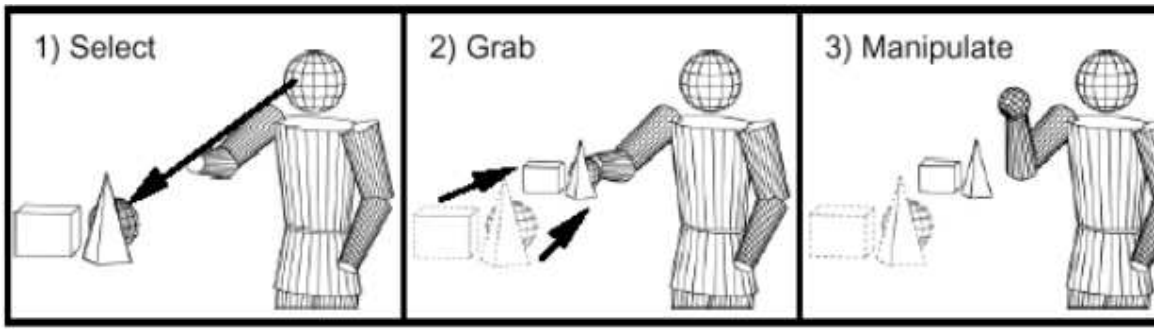


**Figure 4:** *An illustration of the scaled-world grab technique.*

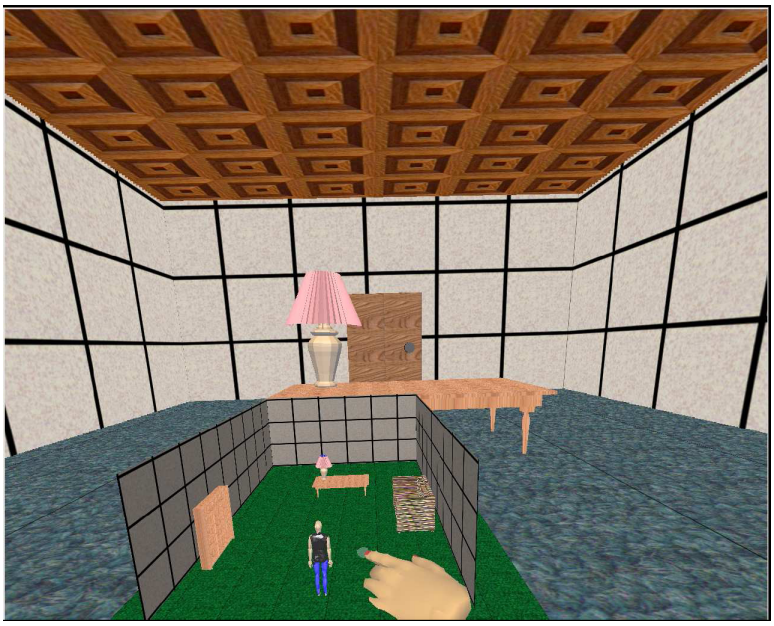

**Figure 5:** *An example of a WIM.*

**World-in-Miniature** The world-in-miniature (WIM) technique uses a small dollhouse version of the world to allow the user to do indirect manipulation of the objects in the environment (see Figure 5). Each of the objects in the WIM are selectable using the simple virtual hand technique, and moving these objects causes the full-scale objects in the world to move in a corresponding way [Stoakley et al. 1995]. The WIM can also be used for navigation by including a representation of the user, in a way similar to the map-based travel technique, but including the 3rd dimension [Pausch et al. 1995].

To implement the WIM technique, first create the WIM. Consider this a room with a table object in it. The WIM is represented as a scaled down version of the room, and is attached to the virtual hand. The table object does not need to be scaled, because it will inherit the scaling from its parent (the WIM room). Thus, the table object can simply be copied within the scene graph.

When an object in the WIM is selected using the simple virtual hand technique, first match this object to the corresponding full-scale object. Keeping a list of pointers to these objects is an efficient way to do this step. The miniature object is attached to the virtual hand, just as in the simple virtual hand technique. While the miniature object is being manipulated, simply copy its position matrix (in its local coordinate system, relative to its parent, the WIM) to the position matrix of the full-scale object. Since we want the full-scale object to have the same position in the full-scale

world coordinate system as the miniature object does in the scaled-down WIM coordinate system, this is all that is necessary to move the full-scale object correctly.

### 2.2.4 System Control

System control provides a mechanism for users to issue a command to either change the mode of interaction or the system state. In order to issue the command, the user has to select an item from a set. System control is a wide-ranging topic, and there are many different techniques to choose from such as the use of graphical menus, voice commands, gestures, and tool selectors. For the most part, these techniques are not difficult to implement, since they mostly involve selection. For example, virtual menu items might be selected using ray-casting. For all of the techniques, good visual feedback is required, since the user needs to know not only what he is selecting, but what will happen when he selects it. In this section, we briefly highlight some of the more common system control techniques.

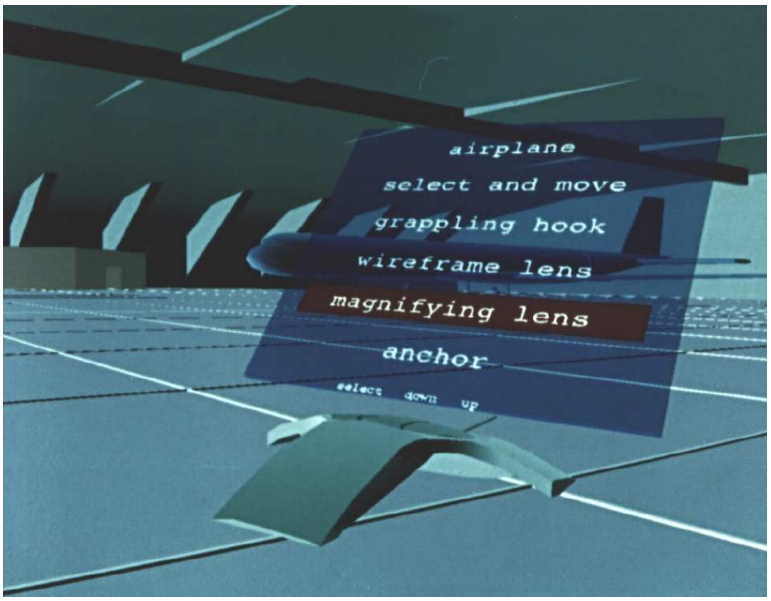


**Figure 6:** *The Virtual Tricorder: an example of a graphical menu with device-centered placement.*

**Graphical Menus** Graphical menus can be seen as the 3D equivalent of 2D menus. Placement influences the access of the menu (correct placement can give a strong spatial reference for retrieval), and the effects of possible occlusion of the field of attention. The paper by Feiner et al. is an important source for placement issues [Feiner et al. 1993]. The authors divided placement into surround-fixed, world-fixed and display-fixed windows. The subdivision of placement can, however, be made more subtle. World-fixed and surround-fixed windows, the term Feiner et al. use to describe menus, can be subdivided into menus which are either freely placed into the world, or connected to an object. Display-fixed windows can be renamed, and made more precise, by referring to their actual reference frame: the body. Body-centered menus, either head referenced or body-referenced, can supply a strong spatial reference frame. One particularly interesting possible effect of body-centered menus is eyes-off usage, in which users can perform system control without having to look at the menu itself. The last reference frame is the group of device-centered menus. Device-centered placement provides the user with a physical reference frame (see Figure 6) [Wloka and Greenfield 1995]. A good example is the placement of menus on a responsive workbench, where menus are often placed at the border of the display device.

We can subdivide graphical menus into hand-oriented menus, converted 2D menus, and 3D widgets. One can identify two major groups of hand-oriented menus. 1DOF menus are menus which use a circular object on which

several items are placed. After initialization, the user can rotate his/her hand along one axis until the desired item on the circular object falls within a selection basket. User performance is highly dependent on hand and wrist physical movement and the primary rotation axis should be carefully chosen. 1DOF menus have been made in several forms, including the ring menu, sundials, spiral menus (a spiral formed ring menu), and a rotary tool chooser. The second group of hand-oriented menus are hand-held-widgets, in which menus are stored at a body-relative position.

The second group is the most often applied group of system control interfaces: converted 2D widgets. These widgets basically function the same as in desktop environments, although one often has to deal with more DOFs when selecting an item in a 2D widget. Popular examples are pull-down menus, pop-up menus, flying widgets, toolbars and sliders.

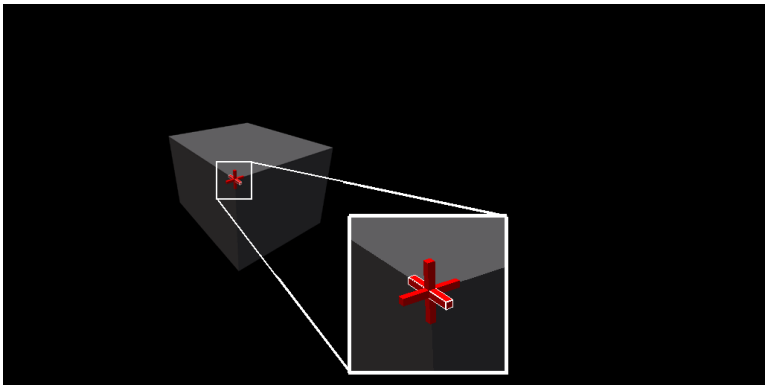

**Figure 7:** *An example of a 3D widget used to scale a geometric object.*

The final group of graphical menus is the group known as 3D widgets [Conner et al. 1992]. In a 3D world, widgets often mean moving system control functionality into the world or onto objects (see Figure 7). This matches closely with the definition of widgets given by Conner et al., ”widgets are the combination of geometry and behavior”. This can also be thought of as ”moving the functionality of a menu onto an object.” A very important issue when using widgets is placement. 3D widgets differ from the previously discussed menu techniques (1DOF and converted 2D menus) in the way the available functions are mapped: most often, the functions are co-located near an object, thereby forming a highly context-sensitive menu.

**Voice Commands** Voice input allows the initialization, selection and issuing of a command. Sometimes, another input stream (like a button press) or a specific voice command is used to allow the actual activation of voice input for system control. The use of voice input as a system control technique can be very powerful: it is hands-free and natural. Still, continuous voice input is tiring, and can not be used in every environment. Furthermore, the voice recognition engine often has a limited vocabulary. In addition, the user first needs to learn the voice commands before they can be applied.

Problems often occur when applications are more complex, and the complete set of voice commands can not be remembered. The structural organization of voice commands is invisible to the user: often no visual representation is coupled to the voice command in order to see the available commands. In order to prevent mode errors, it is often very important to supply the user with some kind of feedback after she has issued a command. This can be achieved by voice output, or by the generation of certain sounds. A very interesting way of supporting the user when interacting with voice and invisible menu structures can be found in telecommunication: using a telephone to access information often poses the same problems to the user as using voice commands in a virtual environment.

**Gestures and Postures** When using gestural interaction, we apply a hand-as-tool metaphor: the hand literally becomes a tool. When applying gestural interaction, the gesture is both the initialization and the issuing of a com-

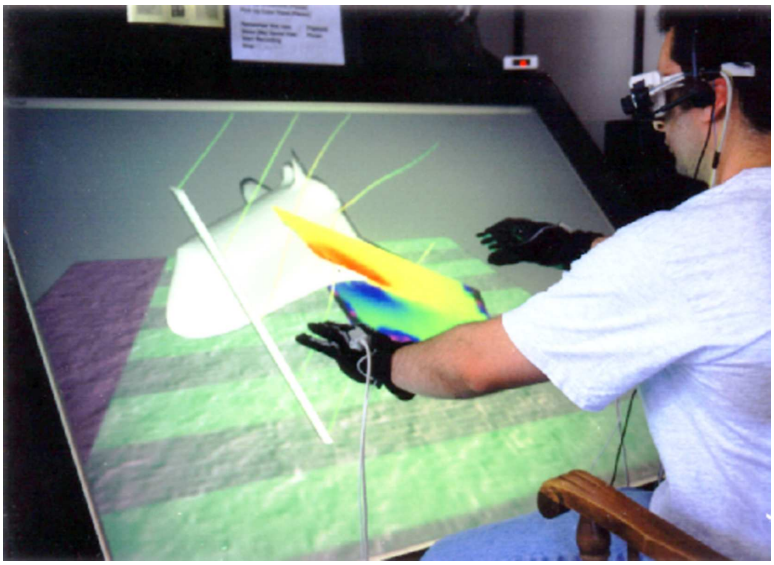

**Figure 8:** *A user interacting with a dataset for visualizing a flow field around a space shuttle. The user simultaneously manipulates the streamlines with his left hand and the shuttle with his right hand while viewing the data in stereo. The user asked for these tools using speech input.*

mand, just as in voice input. When talking about gestural interaction, we refer, in this case, to gestures and postures, not to gestural input with pen-and-tablet or similar metaphors. There is a significant difference between gestures and postures: postures are static movements (like pinching), whereas gestures include a change of position and/or orientation of the hand. A good example of gestures is the usage of sign language.

Gestural interaction can be a very powerful system control technique. However, one problem with gestural interaction is that the user needs to learn all the gestures. Since the user can normally not remember more than about 7 gestures (due to the limited capacity of the working memory), inexperienced users can have significant problems with gestural interaction, especially when the application is more complex and requires a larger amount of gestures. Users often do not have the luxury of referring to a graphical menu when using gestural interaction - the structure underneath the available gestures is completely invisible. In order to make gestural interaction easier to use for a less advanced user, strong feedback, like visual cues after initiation of a command, might be needed. An example of application that used a gestural system control technique is MSVT [LaViola 2000]. This application combined gestures and voice input to create a multimodal interface for exploratory scientific visualization (see Figure 8).

**Tools** We can identify two different kinds of tools, namely physical tools and virtual tools. Physical tools are context-sensitive input devices, which are often referred to as props. A prop is a real-world object which is duplicated in the virtual world. A physical tool might be space multiplexed (the tool only performs one function) or time multiplexed, when the tool performs multiple functions over time (like a normal desktop mouse). One accesses a physical tool by simply reaching for it, or by changing the mode on the input device itself.

Virtual tools are tools which can be best exemplified with a toolbelt (see Figure 9). Users wear a virtual toolbelt around the waist, from which the user can access specific functions by grabbing at particular places on belt, as in the real world. Sometimes, functions on a toolbelt are accessed via the same principles as used with graphical menus, where one should look at the menu itself. The structure of tools is often not complex: as stated before, physical tools are either dedicated devices for one function, or one can access several (but not many) functions with one tool. Sometimes, a physical tool is the display medium for a graphical menu. In this case, it has to be developed in the same way as graphical menus. Virtual tools often use proprioceptive cues for structuring.

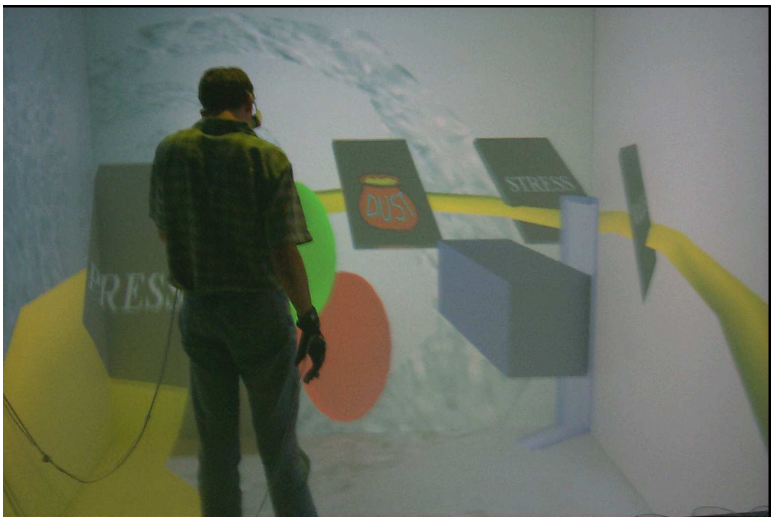


**Figure 9:** *An example of a virtual toolbelt. The user looks down to invoke the belt and can grab tools and use them in the virtual environment.*

It is still unclear how to best design tools for system control. Still, some general design issues can be stated. In the case of physical tools, the form of the tool often strongly communicates the function one can perform with the device, so take care with the form when developing new props. The form of a tool can highly influence the directness and familiarity with the device as well. With respect to virtual tools which can not be used without looking at them, their representation can be very similar to graphical menus.


**Figure 10:** *The Pen and Tablet Metaphor. The image on the left shows the user holding the pen and tablet and the image on the right shows what the user's sees in the virtual world.*

One example of a commonly used physically-based tool is the pen and tablet. Users hold a large plastic tablet on which a (traditional) 2D interface is displayed in the virtual world (see Figure 10). Users are able to use graphical menu techniques and can move objects with a stylus within a window on the tablet. The tablet supplies the user with strong physical cues with respect to the placement of the menu, and allows increased performance due to faster selection of menu items [Bowman et al. 1998].

For the implementation of this technique, the most crucial thing is the registration (correspondence) between the physical and virtual pens and tablets. The tablets, especially, must be the same size and shape so that the edge of the physical tablet, which the user can feel, corresponds to the edge of the virtual tablet as well. In order to make tracking easy, the origin of the tablet model should be located at the point where the tracker is attached to the physical tablet, so that rotations work properly. Even with care, its difficult to do these things exactly right, so a final tip is to

include controls within the program to tweak the positions and/or orientations of the virtual pen and tablet, so that they can be into registration if there's a problem.

Another useful function to implement is the ability for the system to report (for example when a callback function is called) the position of the stylus tip in the tablet coordinate system, rather than in the world or user coordinate systems. This can be used for things like the map-based travel technique described earlier.

### 2.3 3D Spatial Interaction in Video Games


**Figure 11:** *A screen shot of SwordPlay, a video game prototype used to explore how 3D UIs can fit in a gaming environment.*

In the last section, we examined some techniques for performing common tasks with 3D user interfaces in virtual environments. We now switch gears and focus on how this type of interface can be used in the video game domain. One of the major barriers for researchers and developers to utilize 3D UIs in video games in the past has been due to the expense of the hardware (e.g., motion sensing input devices, 3D stereo displays). Although there has been some work in this area, such as SwordPlay (see Figure 11), a focus on 3D UI and gameplay has been minimal. SwordPlay is a game designed to explore what 3D user interfaces might be appropriate in a 3D, immersive gaming environment where two tracked wands are used to invoke a sword and shield and a bow and arrow to fight enemies (players can also cast spells in 3D by drawing with the sword) [Katzourin et al. 2006].

Recently, motion sensing components have gotten much cheaper, resulting in the ability to have these devices available to the mass market. In particular, with the release of the Wii, the latest gaming console from Nintendo, has made one of the most important technological innovations in gaming technology with respect to 3D user interfaces. The key innovation of the Wii is its controller, the Wiimote. This input device not only acts as a gamepad, but makes games accessible to the casual gamer because it can sense 3D motion and also provides researchers, developers, and hobbyists with the ability to explore their own ideas because the device can be connected to a standard computer. Although the Wiimote is a relatively simple device, its is quite challenging to use given its unique configuration. Thus, in this section we explore the intricacies of the Wiimote and present some examples of how it has been used in games research. The ultimate goal of this section is to provide the reader with enough information to get started working with Wiimotes for exploring 3D UIs.

### 2.3.1 Understanding the Wiimote

The Nintendo Wiimote is an example of a spatially convenient device. It provides spatial data, numerous functions and is designed for commodity use. The spatial data of the Wiimote is limited, prohibiting it from extracting a full 6 degrees of freedom (DOF). These limitations also result in the device providing certain types of information based on how it is held and whether it is stationary or moving. The combination of its accelerometers and optical sensor can provide a variety of useful information for creating 3D user interfaces. Some of this information comes directly from the sensors, but other pieces of information come from mathematical derivations.

The Wiimote has a variety of functionality. First, the Wiimote contains Bluetooth for wireless transmission of its data to a computer up to 30ft. This enables it to move freely and be readily repositioned for its intended uses. Second, the Wiimote has several easily manipulatable buttons for discrete event input. Lastly, the Wiimote holds a speaker for audio and rumble feedback. The Wiimote is a commodity device. First, the setup of the device involves placing a sensor bar on a surface and connecting to a computer through Bluetooth. Second, the Wiimote consumes low amounts of energy, enabling its batteries to fit into a small, lightweight, easily held package. Lastly, the mass production of the Wiimote enables it to currently retail for 40 US dollars.

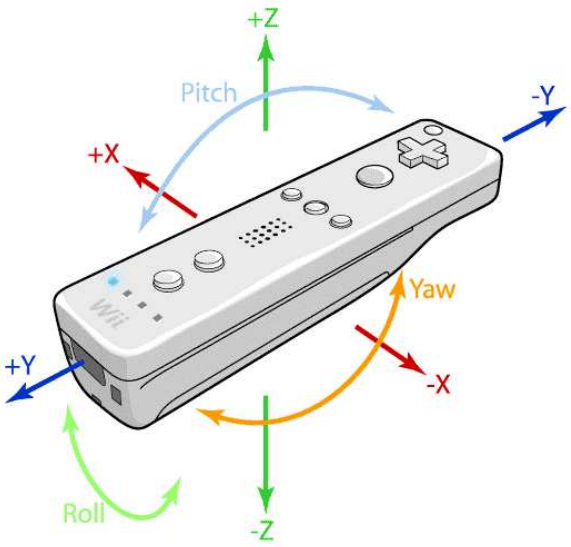


**Figure 12:** *The Wiimote coordinate system.*

**Frames of Reference** Using the Wiimote entails multiple frames of reference. Figure 12 shows the Wiimote's frame of reference (FOR) and data reported by the Wiimote is in this reference frame. The second is the Earth's FOR. The Earth is important as its gravity is detected by the Wiimote's accelerometers (see 3-Axis Accelerometer below). The final FOR is the Wiimote's relationship to the sensor bar (see The Sensor Bar below). To clarify these FOR, consider the following examples. Holding a Wiimote level (+Z up in the Wiimote and Earth's FOR) and facing the sensor bar, which is placed in front of the Wiimote, the user moves the Wiimote towards the sensor bar. This results in acceleration reported in the Y-axis of both the Earth and Wiimote FOR and the sensor bar reporting a shrinking distance between the Wiimote and sensor bar. Now, pointing the Wiimote down towards the earth and again moving the Wiimote forward, the acceleration in the Earth's FOR is unchanged in the Z-axis but is in the Z-axis of the Wiimote's FOR (the Y-axis in the Earth's FOR). Regarding the sensor bar's FOR, as the Wiimote is rotated, it does not see the sensor bar so there is no FOR. In this last example, if we place the sensor bar on the floor and repeated the forward movement, then the sensor bar would report the Wiimote moved up in the Y-axis. The utility of each FOR will become apparent in the following sections.

**The Sensor Bar** There are two primary spatial inputs that can be utilized with the Wiimote. The first input is the sensor bar connection (SBC), which results from the Wiimote's infrared optical sensor pointing into the distance at a sensor bar, easily identified by its 5 LEDs on each side (see Figure 13). The LEDs have a known width between them and the LEDs are spread in a slight arch with the furthest pointed away from the center and the closest pointed inwards. This spread helps to improve the SBC's range, effectively operating the Wiimote up to 16ft away from the sensor bar.

**Figure 13:** *The Wiimote sensor bar has two groups of IR LEDs at fixed widths.*

When the Wiimote is pointed at the sensor bar, it picks up two points $P_L = (x_L, y_L)$ and $P_R = (x_R, y_R)$ from the LED arrays. The midpoint between these $P_L$ and $P_R$ can easily be calculated and used as a 2D cursor or pointer on the display. In addition, if the Wiimote is rotated about the Y-axis, we can calculate the roll of the device with respect to the X-axis using

$$roll = \arccos\left(\mathbf{x} \cdot \frac{\mathbf{v}}{\|\mathbf{v}\|}\right) \tag{4}$$

where $\mathbf{x} = (1, 0)$ and $\mathbf{v} = P_L - P_R$.

Combining information from the sensor bar and the Wiimote's optical sensor also make it possible to determine how far the Wiimote is from the sensor bar using triangulation. The distance $d$ between the sensor bar and Wiimote is calculated using

$$d = \frac{w/2}{\tan(\theta/2)} \tag{5}$$

$$w = \frac{m \cdot w_{img}}{m_{img}} \tag{6}$$

$$m_{img} = \sqrt{(x_L - x_R)^2 + (y_L - y_R)^2} \tag{7}$$

where $\theta$ is the optical sensor's viewing angle, $m$ is the distance between the sensor bar's left and right LEDs, $w_{img}$ is the width of the image taken from the optical sensor, and $m_{img}$ is the distance between $P_L$ and $P_R$ taken from the optical sensor's image. Note that $\theta$, $w_{img}$, and $m$ are all constants. This calculation only works when the Wiimote device is pointing directly at the sensor bar (orthogonally). When the Wiimote is off-axis from the sensor bar, more information is needed to find depth. We can utilize the relative sizes of the points on the optical sensor's image to calculate depth in this case. To find $d$ in this case, we compute the distances corresponding to the the left and right points on the optical sensor's image

$$d_L = \frac{w_L/2}{\tan(\theta/2)} \tag{8}$$

$$d_R = \frac{w_R/2}{\tan(\theta/2)} \tag{9}$$

using

$$w_L = \frac{w_{img} \cdot diam_{LED}}{diam_L} \tag{10}$$

$$w_R = \frac{w_{img} \cdot diam_{LED}}{diam_R} \tag{11}$$

where $diam_{LED}$ is the diameter of the actual LED marker from the sensor bar and $diam_L$ and $diam_R$ are the diameters of the points found on the optical sensor's image. With $d_L$ and $d_R$, we can then calculate Wiimote's distance to the sensor bar as

$$d = \sqrt{d_L^2 + (m/2)^2 - 2d_L(m/2)^2 \cos(\phi)} \tag{12}$$

where

$$\cos(\phi) = \frac{d_L^2 m^2 - d_R^2}{2md_L}. \tag{13}$$

Note that with $d$, $d_L$, and $m$ we can also find the angular position of the Wiimote with respect to the sensor bar, calculated as

$$\alpha = \arccos\left(\frac{d^2 m^2 - d_L^2}{md_L}\right). \tag{14}$$

**3-Axis Accelerometer** The second Wiimote input is the device's 3-axis accelerometer. The accelerometer reports acceleration data in the device's x, y and z directions, expressed conveniently in g's. This is common of many devices employing 3-axis accelerometers such as the cell phones like the iPhone, laptops and camcorders. With this information, the Wiimote is able to sense motion, reporting values that are a blend of accelerations exerted by the user and gravity. As the gravity vector is constantly oriented towards the Earth (or $(0, 0, 1)$ in Earth's FOR), the gravity vector can be used to discover part of the Wiimote's orientation in terms of earth's frame of reference using

$$pitch = \arctan\left(\frac{a_z}{a_y}\right) \tag{15}$$

$$roll = \arctan\left(\frac{a_z}{a_x}\right). \tag{16}$$

Two issues make determining the Wiimote's orientation from the accelerometer data a possible but problematic computation. First, yaw in the Earth's FOR cannot be determined as the Earth's gravity vector aligns with the Y-axis (the same is true for both pitch and roll when their axes aligns with the Earth's FOR). Second, the gravity vector is only known when the Wiimote is held steady; otherwise it is confounded by other accelerations. This limits its use as an orientation tracker while it is under motion.

Determining the acceleration in the Wiimote's frame of reference is problematic as well. Unfortunately, the gravity vector confounds the reported acceleration of the Wiimote. So, the gravity vector must first be removed from the reported acceleration before the acceleration in the Wiimote's frame of reference can be obtained. Unfortunately,

this requires knowing the orientation of the gravity vector which means either: (1) the Wiimote must be at rest, (2) assumptions are made about the Wiimote's orientation or (3) orientation is obtained externally by the SBC or a gyroscope (see Gyroscopes below). With the orientation, the gravity vector can be subtracted from the current acceleration data to obtain the Wiimote's acceleration.

**"Waggling" Movements** Cheating motions! Or, less fatiguing and comfortable movements, depending on your perspective, are a side effect of the limitations of the Wiimote's input. While game designers may intend for games to encourage exercise, breaking the stereotype of the lazy video game player, the inability to detect actual position change enables users to make only small or limited little "waggling" motions that are interpreted as full movement. The result is boxing games played by tapping the Wiimote, tennis games played with wrist flicks and many games winnable by simply moving the device around in random directions. While still fun for gamers, this limits the application of the Wiimote for use in exercise and health gaming along with 3D UIs unless better hardware and data interpretation methods are employed.

**Gyroscopes** Gyroscopes report orientation. While mechanical gyroscopes are typically too large and expensive for a Wiimote, MEMS gyroscopes are cheap, low power and fairly accurate. As such, a Wiimote with a gyroscope could provide orientation information, alleviating many of the Wiimote's issues. This is not just an academic idea, but is currently being employed by Nintendo in the production of the Wii Motion Plus add-on to the Wiimote (released by the time of this lecture's printing). The Wii Motion Plus is a two-axis gyroscope plug-in to the Wiimote to be released in 2009. With this, Nintendo is attempting to achieve a closer match of the Wii's motion to the real world. The accuracy of this device is to be determined.

**Compensation by "Story"** When possible, the easiest means of compensating for limitations in input hardware is through the use of story. By this we mean that by careful manipulation of the user's tasks and their goals, the shortcomings of the hardware can be avoided. This is commonly used in Wii gaming, where accuracy is second to enjoyment and playability. In these cases, the inherent drift can be overridden by requiring the user to return to a neutral position, Wiimote orientation can be assumed by telling them how to hold the Wiimote and SBCs can be created by requiring the user to point at an on-screen button to begin a task. In the same way, story can engage the user, instructing them to freeze their hand at a fixed orientation for a static reading the gravity vector. After this, acceleration can be transformed into position information in relation to the gravity vector and any drift can be accounted for at the end of the trial. For example, the We Cheer game instructs the user to hold the Wiimote like they were holding pom-poms, WarioWare instructs how to hold the Wiimote for each mini-game and Wii Sports Boxing requires raising the hands in preparation for a fight. While these are gaming "stories", researchers commonly guide participants, instructing them to enter start positions before the beginning of a trial. For example, in studies of selection techniques, it is common to have users select an object to begin a trial. By proper use of story, many of the limitations of the hardware can be successfully hidden from the user.

### 2.3.2 Case Studies

The Wiimote has been used in a variety of different ways. Of course, Johnny Chung Lee has done a variety of interesting projects and is probably the most famous Wiimote hacker [Lee 2008]. However, there are many other researchers who are using Wiimotes for research in spatial 3D interfaces. Typically, there are two main approaches people use. The first is to actually wear the sensor bar and mount the Wiimote in some stationary position, using it as a camera. The second approach is to simply hold the Wiimote or attach it to the body. This approach makes more use of the accelerometer data than the image sensor data. In this section, we highlight some interesting research that focuses on using Wiimotes to create better gameplay experiences.

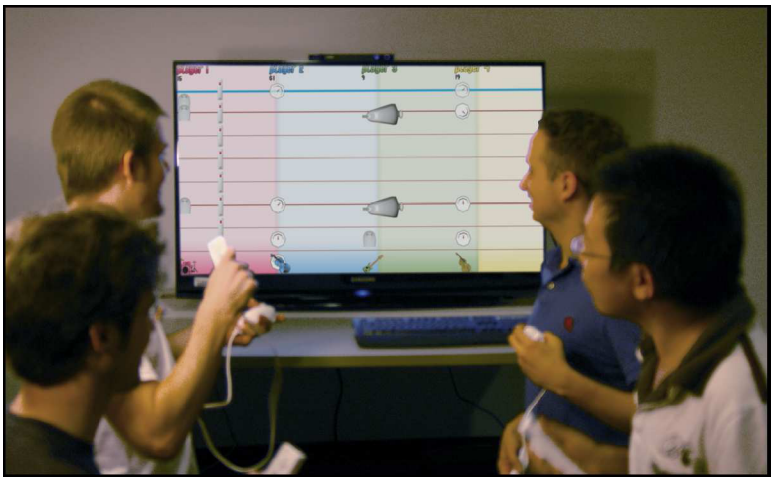

**Figure 14:** *Users playing One Man Band.*

**One Man Band** One Man Band (see Figure 14) is a prototype video game for musical expression that uses novel 3D spatial interaction techniques using accelerometer-based motion controllers [Bott et al. 2009]. One Man Band provides users with 3D gestural interfaces to control both the timing and sound of the music played, with both single and collaborative player modes. The current system supports the guitar, violin, bass, drums, trombone, and therein. It also has a multi-instrument musical interface called the MIMI. The idea behind the MIMI is that game players might want to quickly and easily transition from one musical instrument to another without any mode switching. The MIMI uses heuristics recognition and exponential smoothing to detect 5 different instruments. A study was recently conducted comparing One Man Band and Wii Music and the results showed users significantly preferred One Man Band.

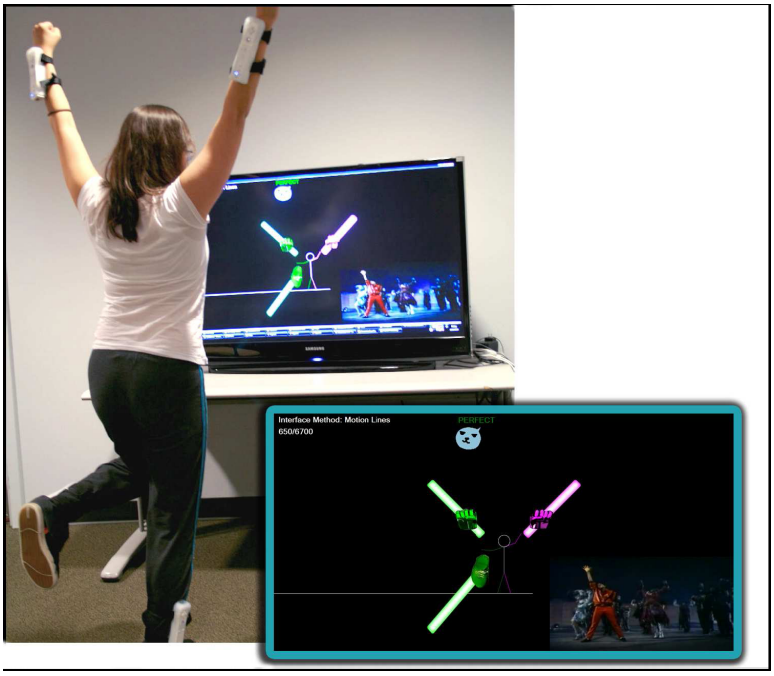


**Figure 15:** *A user playing RealDance.*

**RealDance** Real Dance (see Figure 15) is a game prototype that is exploring more natural, full body interfaces for dancing games [Charbonneau et al. 2009]. In the game, users wear four Wiimotes attached to their wrists and ankles using velcro strips. This provides an untethered experience, meaning that the user does not need to stand in one place or position. Another goal of Real Dance is to explore how to increase the number of recognizable dance movements. The current prototype detects kicks, stomps, punches, and static poses. It also employs a visual interface for teaching users to perform the various dances. The visual interface provides either a timeline with icons, motion lines, or beat circles as well as score feedback with avatars and an animated instructor figure.

**Figure 16:** *A user playing World of Warcraft using body-based controls to navigate.*

**Figure 17:** *A user wears the sensor bar in Wiisoccer.*

**WoW Navigation** World of Warcraft has a complex interface and work is being done to determine how body-based interaction can be used to complement and reduce this complexity [Silva and Bowman 2009]. The idea is to using body based controls to offload keyboard and mouse navigation, helping players to concentrate on other tasks (see Figure 16). In this configuration, the user wears a modified sensor bar and mounts a Wiimote on the ceiling, using the device as a camera. Navigation is based on a leaning metaphor. Starting from a neutral body position, a

small amount of forward or backward movement makes the character walk, and leaning farther forward makes the character run (rate control). There is a dead zone surrounding the neutral point in which the character stands still. Leaning to the side rotates the character, with the amount of rotation proportional to the distance the player leaned (position control). Note that a foot pedal is used to activate and deactivate movement. Preliminary experimentation has shown that body-based interaction in addition to keyboard and mouse can help players perform more tasks at the same time and can be especially attractive and helpful to new players.

**Exergaming** The focus of this project, developed at Brown University under the direction of Chad Jenkins, is on exergaming, an important and up-and-coming research area that examines how to build effective exercise-based games. Wiisoccer is a game that uses natural locomotion to move players on a soccer field. The key innovation is that the IR sensor bar is attached to the users leg (see Figure 17) and a Wiimote is used as a camera and placed on the side of the player. The Wiimote detects the players running motion as well as kicks and passes.

## 2.4 Conclusions

The divide between video games and 3D user interfaces is shrinking. As more affordable, commodity 3D spatial hardware becomes available, we will see an increased use of 3D UIs for more expressive gameplay. In this lecture, we have examined several techniques that are used for common tasks, such as navigation, selection, manipulation, and system control, in 3D virtual environments. These techniques are directly applicable to the video game domain.

The Wiimote is a powerful input device that is not just for playing games with the Nintendo Wii console. It is a great device for exploring 3D spatial interfaces because it is inexpensive compared to other tracking systems, it works with a PC, and provides motion and pointing data. Although the device is not perfect because of the ambiguity and constrained input problem, it can still be used to implement a variety of 3D user interface techniques. The Wiimote continues to evolve and by the time this course has been given, Nintendo will have released the Wii Motion Plus, making the Wiimote more expressive, given the orientation sensing from its gyroscopes. Note that the Wiimote will never be ideal for implementing the 3D UI techniques discussed in this lecture until it can sense a true 6 DOF (position and orientation). Other commodity devices are on the horizon, such as Sixense's TruMotion device, that will provide this capability making the the integration of 3D UIs into video games even easier. Of course, there is still a lot of research to be done in finding the best ways of using these devices and 3D UIs in the video game domain.

## Acknowledgements

A special thanks to Doug Bowman, Ernst Kruijff, Ivan Poupyrev, and Chad Wingrave for assisting in the development of the material for this lecture.

# 3 Sketch-Based Interfaces for Computer Graphics

Creation and control of three-dimensional (3D) models for computer graphics authoring are still very difficult because traditional interfaces for 3D modeling programs have their origins in professional drafting. The user places vertices in 3D space by specifying $x$-, $y$-, and $z$-coordinates in a three-view interface and then creates faces by connecting these vertices. Alternatively, the user starts with a simple primitive, such as a sphere or cylinder, and modifies it by editing individual vertices and edges. Many editing tools (e.g., free-form deformation and Boolean operations among solids) are also available for designing complicated shapes from simple primitives. Authoring of animation is also difficult. In the typical user interface, the user specifies individual key frames, which is very laborious. Although these interfaces may be appropriate for trained users designing precise models and animations, they are too difficult for first-time users to quickly generate meaningful models and animations.

The sketching interface is emerging as an alternative modeling and control method taking its inspiration from the observation of human-to-human communication. People quickly sketch their ideas on a paper with a pen for communication and self-exploration. Sketching interfaces try to introduce this type of fluid interaction into creation and control of 3D models. The user simply draws two-dimensional (2D) lines on the screen; the system infers additional information from the user input and presents the result. In the case of 3D modeling, the system infers missing depth information. In the case of 3D animation, the system infers detailed motion from simple input.

Two key issues exist in designing effective sketching systems. One is the design of algorithms to infer the missing information from the user input. Inference algorithms generally rely on special domain knowledge to achieve this. In addition, an infinite number of possible 3D configurations match the given 2D input, so the 2D to 3D problem is not well-defined in general cases. However, specific domains impose certain constraints on the possible 3D configuration and the system can return reasonable results by searching the problem space within these constraints. The other is the design of a user interface for disambiguation. Sketching is inherently ambiguous and having the computer return the correct answer with 100% certainty is generally difficult. Therefore, one must allow the user to modify the result or request other guesses.

This course note introduces several sketching systems for computer graphics authoring. Individual systems address specific problems and propose specific techniques independently, and generalizing these systems is therefore difficult. However, they all attempt to address the problem of inferring missing information using domain knowledge and solving the ambiguity problem by providing special disambiguation interfaces. We hope that readers will obtain some useful information to aid in resolving their own problems from these example systems. Please note that this course note is not a comprehensive survey of the field. The goal is to introduce key ideas behind sketching systems and the majority of this note is based on our own work.

We focus mainly on sketching interfaces for 3D computer graphics authoring, especially shape modeling, deformation, and animation. Many other interesting systems use sketching, such as user-guided image editing, but these are beyond the scope of this note.

## 3.1 2D Drawing - Interactive Beautification

The Pegasus system [Igarashi et al. 1997] allows the user to construct precise illustrations as shown in Fig. 18 without using complicated editing commands. The concept is to automate complicated drawing operations by having the computer infer possible geometric constraints and the user's next steps from the user's free-form strokes. Interactive beautification receives the user's free stroke input and beautifies it considering possible geometric constraints among line segments by generating multiple alternatives to prevent recognition errors. Predictive drawing foretells the user's next drawing operation based on the spatial relationships among existing segments on the screen. A prototype system is implemented as a Java$^{TM}$ program, and our preliminary user study showed promising results.

One notable feature of this system is that it provides a good disambiguation method. When the system shows the beautification result, it also presents secondary candidates to the user in addition to the default result (Fig. 19). If the

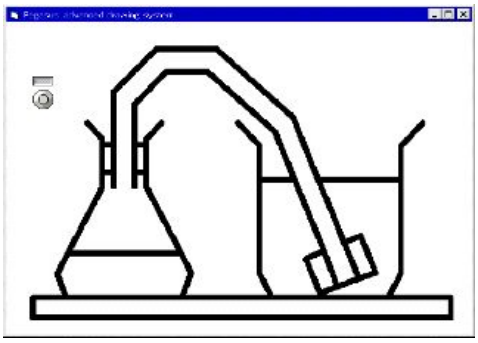

**Figure 18:** *Screenshot of interactive beautification.*

default result is not satisfactory, the user can quickly switch to a secondary result with simple clicking. We found that single answer fails too often and secondary candidates are essential for making this system usable.

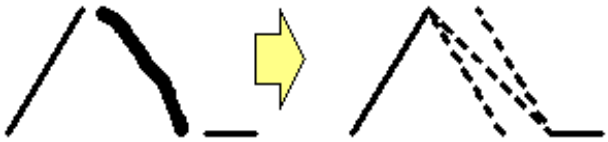

**Figure 19:** *Beautification of an input stroke and multiple candidates.*

## 3.2 3D Shape Modeling

### 3.2.1 Architecture Models

**SKETCH:** The SKETCH system introduced the gesture-based 3D scene construction technique [Zeleznik et al. 1996]. A simple gesture creates a 3D primitive object and places it in a 3D scene (Fig. 20). The system calculates the object's position based on the assumption that every object in the scene should be on some other object. For example, when the user draws three lines requesting a box in the 3D scene, the system puts the box on top of the existing box. In addition, the plate on the floor is automatically lifted in 3D space when the user draws a leg under the plate, without changing the 2D appearance in the 2D window. As a result of these implicit placement rules, the user can construct 3D scenes without using 2D widgets and can concentrate on the task (constructing a 3D scene) instead of spending time interacting with nested menus and commands.

**Chateau:** The Chateau system [Igarashi and Hugues 2001] introduced an interface for 3D drawings that improves the usability of gestural interfaces and augments typical command-based modeling systems. In their suggestive interface, the user gives hints about a desired operation to the system by highlighting related geometric components in the scene. The system then infers possible operations based on the hints and presents the results of these operations as small thumbnails (Fig. 21). The user completes the editing operation simply by clicking on the desired thumbnail. The hinting mechanism allows the user to specify geometric relations among graphical components in the scene, and the multiple thumbnail suggestions make it possible to define many operations with relatively few distinct hint patterns. The suggestive interface system is implemented as a set of suggestion engines working in parallel and can be extended easily by adding customized engines. The prototype 3D drawing system shows that a suggestive interface can effectively support the construction of various 3D drawings (Fig. 21).

### 3.2.2 Free-form Models

**Teddy:** The Teddy system [Igarashi et al. 1999] is a sketching scheme for constructing free-form models. It allows

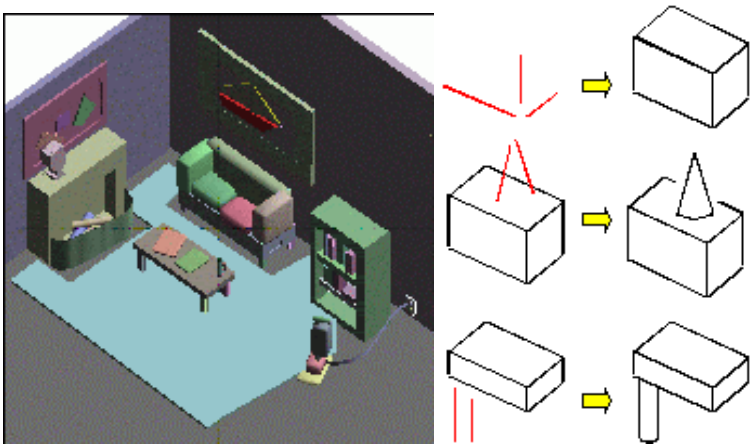

**Figure 20:** *The SKETCH system.*

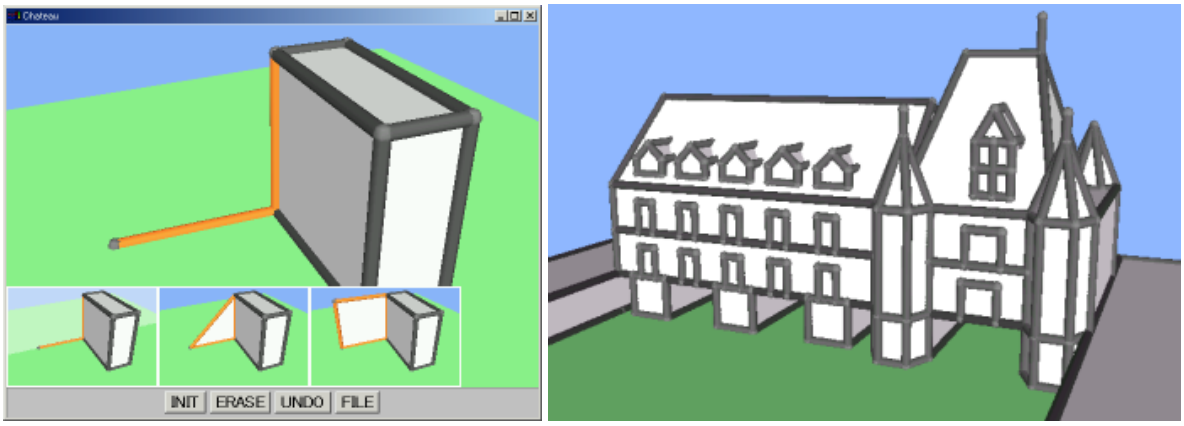


**Figure 21:** *The Chateau system.*

the user to quickly generate interesting 3D free-form models, such as creating a teddy bear simply by drawing the silhouette of the desired shape. Fig. 22 shows an example modeling sequence in Teddy. The user's strokes are shown in red; everything else is inferred and drawn by the system. The user first draws the silhouette of the base primitive, and the system generates the corresponding 3D geometry. The user then draws a stroke across the model, and the system cuts the model at the line. The user can also add parts to the base model by drawing two strokes. Fig. 23 shows several 3D models created using the system.

This type of quick sketching system can be useful for nontraditional 3D graphics applications. First, this tool can be used by nonexperts, including children, who wish to play with 3D graphics for fun. Teddy is already used in several commercial video games to permit players to create their own characters. Second, using this tool can be a useful way for experts to express their ideas quickly in the early design phases. A commercial 3D modeling package already includes an extension of Teddy as a plug-in for generating rough sketches. Finally, and most importantly, this tool will be useful for communicating 3D concepts face to face. In the classroom, the teacher can quickly draw a model of a bacterium and show the cross section to explain the internal structure. In a hospital, a medical doctor can draw a model of the stomach and explain the current status of a stomach disease to the patient. Fig. 24 shows an example in which a high school teacher used the system to teach the concept of contour lines in geography class.

The original Teddy system used a single low-polygonal mesh as a surface representation. It only supported a surface with spherical topology and lacked the ability to edit individual parts afterward. Several other attempts have been made to address these problems using different geometry representations (Fig. 25). Owada et al. [2003] developed

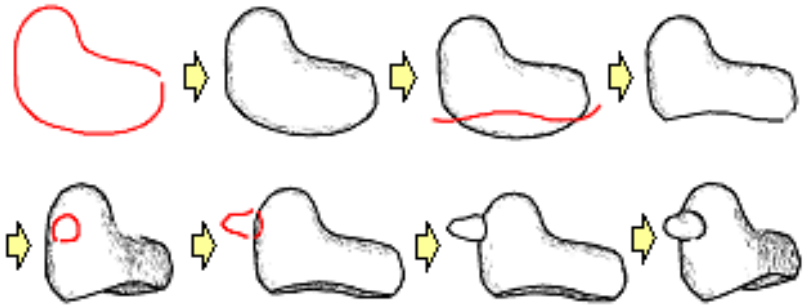

**Figure 22:** *Modeling session in Teddy. The user can create a 3D model using simple sketching operations.*

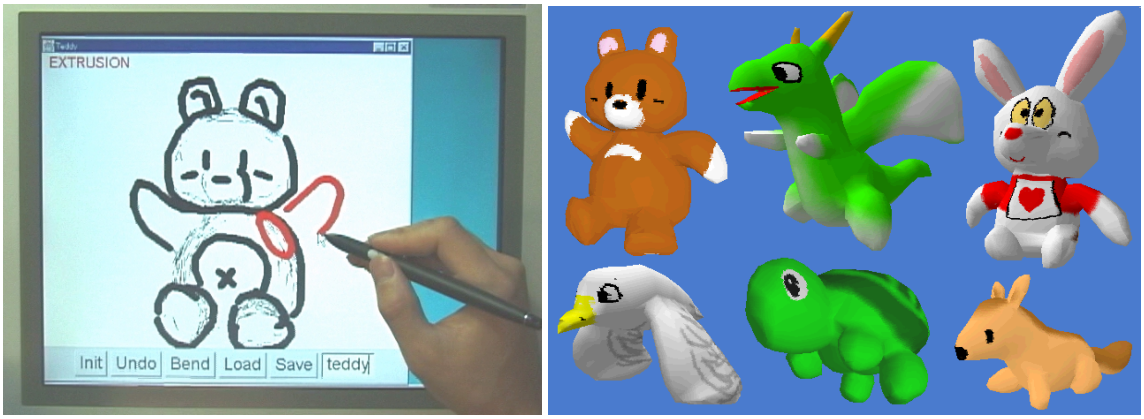


**Figure 23:** *Screenshot of Teddy and sample 3D models created using the system.*

a system, Vteddy, based on voxel representation to handle topological changes. This system is useful for creating a model with holes and loops, such as a heart. Schmidt et al. [2005] developed a system called ShapeShop, which represents a model as a smooth integration of blob primitives and supports editing of these primitives afterward. For example, the user can adjust the location of a body part while preserving smooth connection between the body parts.

**FiberMesh:** Nealen et al. [2007] developed the FiberMesh system to create a model with a fair surface design and allow flexible editing (Fig. 26). It also uses polygonal mesh representation, but computes it via optimization that minimizes curvature variation. The user first creates a rough 3D model using a sketching interface. Unlike other sketching systems, the user-drawn strokes remain on the model surface and serve as handles for controlling the geometry. The user can add, remove, and deform these control curves easily, as if working with a 2D line drawing. The curves can have an arbitrary topology and need not be connected to each other. For a given set of curves,

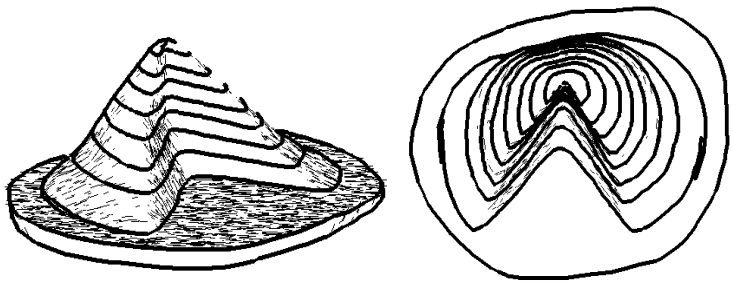

**Figure 24:** *Teaching the concept of contour lines using Teddy.*

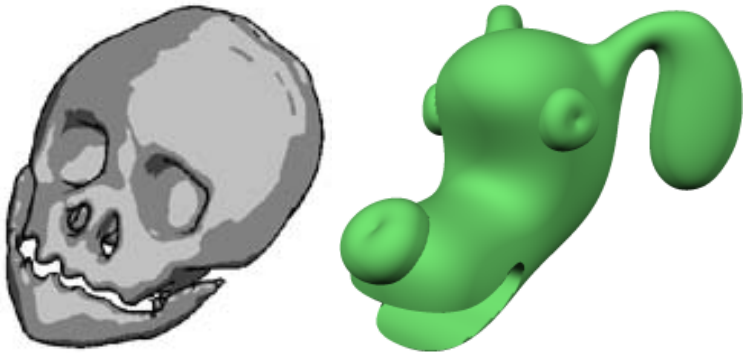

**Figure 25:** *Vteddy [Owada et al. 2003] uses voxel and ShapeShop [Schmidt et al. 2005] uses blob tree representation.*

the system automatically constructs a smooth surface embedding by applying functional optimization. The system provides real-time algorithms for both control curve deformation and subsequent surface optimization.

**Figure 26:** *The strokes remain on the model surface as control curves, which the user can pull to edit the surface.*

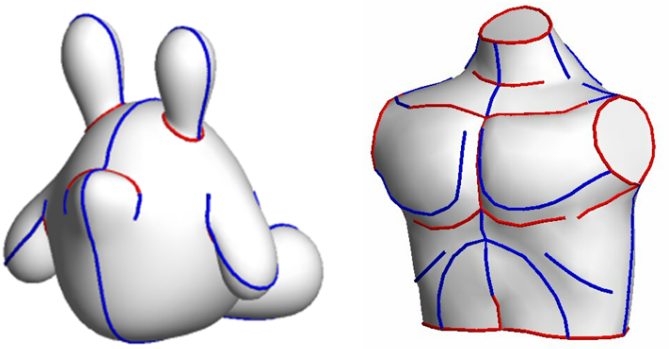

**Figure 27:** *3D models created using FiberMesh. Blue represents smooth control curves and red represents sharp control curves.*

**Plushie:** Mori et al. [2007] developed a sketch-based modeling system for the design of physical plush toys, called Plushie. The systems introduced above were all designed for virtual representations and did not consider any physical constraints. In contrast, a plush toy is a real physical object and cannot take an arbitrary form. Therefore, the Plushie system considers the physical constraints in the modeling process to facilitate the design of real plush toys. The user interface is similar to that of the original Teddy system [Igarashi et al. 1999]. The user interactively draws a sketch on the screen, and the system automatically generates a 3D plush toy model (Fig. 28 and Fig. 29). In addition, the system simultaneously generates a 2D cloth pattern corresponding to the 3D geometry so that the user can create a physical plush toy by cutting the cloth according to the generated pattern. Internally, the system first generates a 2D cloth pattern and then runs a simple physical simulation to predict the 3D shape of the resulting toy. In this way, even young children can design their own plush toys simply by sketching.

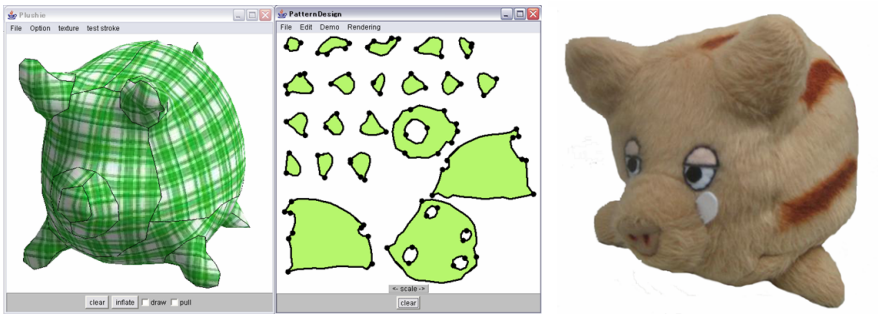

**Figure 28:** *Screenshot of the Plushie system, and a plush toy designed using the system.*

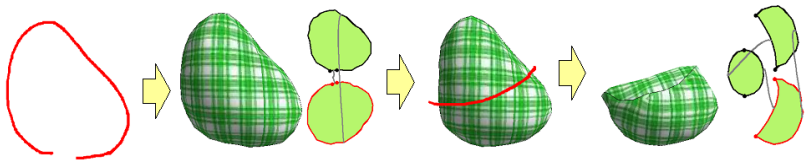

**Figure 29:** *The system automatically constructs the 3D geometry and 2D cloth pattern for the given user input.*

### 3.2.3 Deformation Techniques

Shape deformation is also a fundamental operation in shape modeling, which is necessary for both refining the static model and creating an animation. Traditional deformation methods rely mostly on the individual positioning of control vertices and achieving global deformation using these methods is a laborious process. Here, we introduce two complementary methods for sketch-based shape deformation.

**Sketching Skeleton:** Kho et al. [2005] presented a method for deforming a 3D shape by drawing a skeleton. The user first draws a reference stroke on the 3D model and then draws a target stroke. The part of the 3D model near the reference stroke is then deformed so that it matches the target stroke (Fig. 30). The basic algorithm is similar to standard blend-shape skinning. The system computes the local coordinates of mesh vertices relative to the reference stroke and then computes the new global coordinates by applying the local coordinates to the target stroke. The authors also presented a method for twisting the model around the sketched axis.


**Figure 30:** *Deformation via skeleton sketching.*

**Sketching Silhouette:** Another method for sketch-based shape deformation is to sketch the target silhouette [Nealen et al. 2005]. The user draws a curve indicating the desired local silhouette and the system deforms the surface near the curve so that the new silhouette matches the user-drawn silhouette. They use a detail-preserving shape deformation method in which the system simultaneously minimizes distance to the target silhouette and the distortion of local surface details. This makes it possible for the user to obtain reasonable results even with a very approximate silhouette sketch (Fig. 31, right).

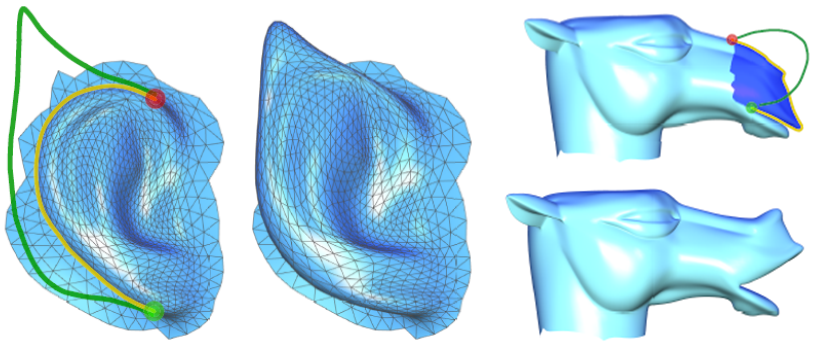

**Figure 31:** *Deformation via silhouette sketching.*

## 3.3 Animations

This section introduces several animation authoring methods. Animation is defined by a collection of deforming 3D models and specifying individual frames is a laborious procedure. The following systems introduce several ways for the user to specify reasonably complicated motions using simple input, such as sketching and direct performance.

### 3.3.1 Articulated Pose Control

Davis et al. [2003] introduced a system for authoring figure animation via simple stick figure drawing. The user draws the 2D projection of the desired pose using a few lines and the system automatically infers the depth of each joint to construct a 3D representation. The system uses domain knowledge about the human body to successfully infer the joint angles. However, reaching a single solution is still difficult and the system asks the user to select the appropriate solution by presenting multiple possibilities. After specifying several key poses, the system then synthesizes an animation sequence by applying optimization.

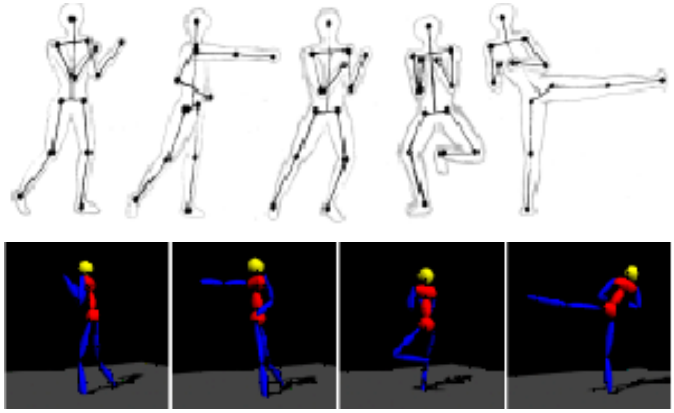

**Figure 32:** *The user draws 2D stick figures and the system generates 3D character animation [Davis et al. 2003].*

### 3.3.2 Motion Doodles

Thorne et al. [2004] developed a system in which the user sketches a single path in 2D space representing the desired trajectory and the system automatically makes an animation sequence, such as walking, running, and jumping. The sketched path can also represent textures of the motion. For example, the user can specify stomping by a zigzag stroke and a double-flip jump by drawing a curve with a double loop in the middle. They also extended the technique to 3D space. Basic motions are crafted by hand in advance and the system adjusts and combines them to produce

the final motion.

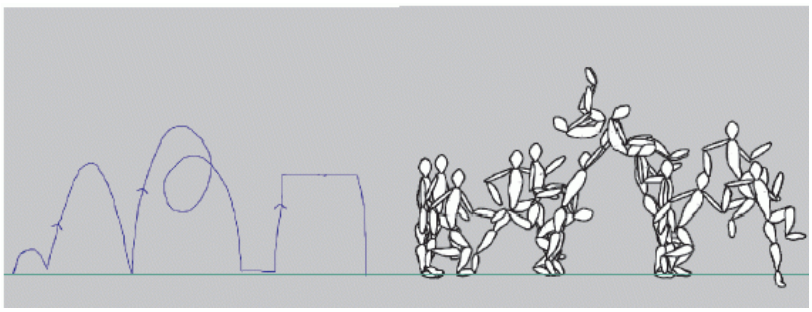

**Figure 33:** *Motion Doodles [Thorne et al. 2004]. The user sketches the trajectory and the system makes an animation.*

### 3.3.3 Animation by Performance

**Spatial Key Framing:** One way to design a motion is through live demonstration in which the user moves the target character in real time and the system records the motion, e.g., a dancing teddy bear in front of a video camera. This can be a much more intuitive interface for inexperienced users to create an animation. However, moving a character with many joints is difficult using a standard input device such as a mouse. Demonstrating the motion of each joint one by one is possible, but synchronizing individual motions is not easy.

The spatial key framing method was proposed to address this problem [Igarashi et al. 2005b]. The user first sets a group of key poses in the 3D or 2D space (a pose is associated with a position in space). The user then moves the cursor in the 3D or 2D space, and the system sets the character pose by blending the key poses around the cursor position (Fig. 34). In this way, the user can design interesting whole-body character motion, such as juggling and dancing, by recording simple cursor movements.

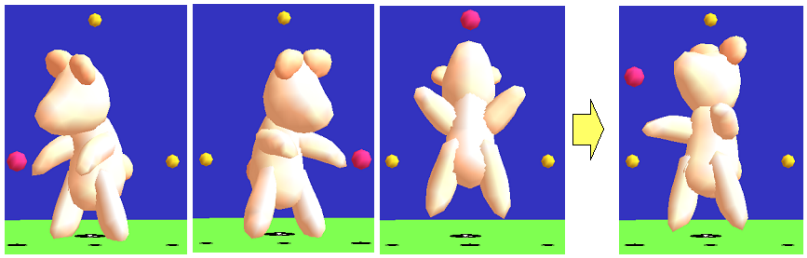

**Figure 34:** *Spatial key framing. The user specifies three key poses (left) and then freely controls the character by dragging the red ball (right).*

This technique is particularly suitable for defining expressive motions, such as a show of joy or sadness. The resulting motion is much more alive than motion generated by traditional key framing because the operator's natural hand movement appears directly in the motion. However, motion dominated by physical factors, such as jumping and running, is better supported by physical movement-based approaches.

**As-Rigid-as-Possible Shape Manipulation:** In the real world, one can hold an object such as a teddy bear with two hands and freely manipulate it by moving, rotating, stretching, squashing, and bending it. However, this is not easy to do on a computer. Standard 2D drawing programs provide poor support for such shape manipulation and offer only simple editing operations, such as scaling and rotation. Not only do these operations require a complicated combination of separate tools, but the result does not feel like manipulating a physical entity.

The as-rigid-as-possible shape manipulation technique [Igarashi et al. 2005a] was developed to address this problem. Using this method, the user can select arbitrary points as handles on a 2D shape and freely manipulate the shape by moving the handles (Fig. 35). The user can relocate the shape by setting a single handle and moving it, and can rotate, stretch, and squash the shape using two handles. Furthermore, the user can swing the head or stretch an arm simply by setting handles on the corresponding positions. The shape deforms naturally in response to the user input, providing the feeling of manipulating a physical object.

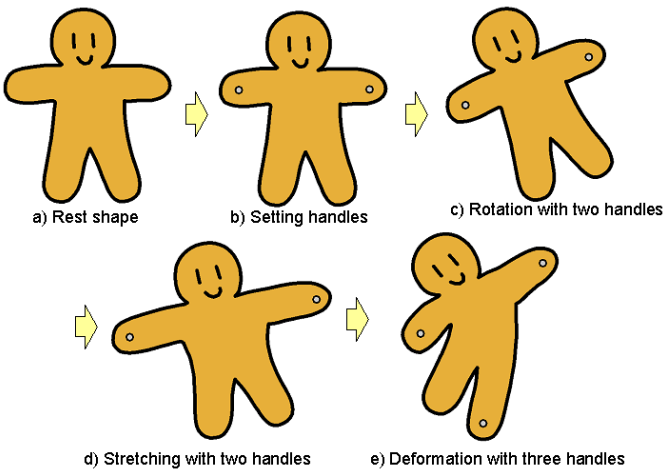


**Figure 35:** *As-rigid-as-possible shape manipulation. The user places handles on the drawing and moves them to manipulate it.*

This technique is particularly useful for creating 2D animations. Traditionally, generating many slightly different drawings is necessary to create an animation. However, in our system, one can create an interesting animation simply by drawing a character and recording the manipulation process. Using a multi-touch input device [Rekimoto 2002], the user can directly grab a character using both hands and manipulate it to create an animation (Fig. 36). We tested this system with children and found that even elementary school students could generate reasonable animations quickly.

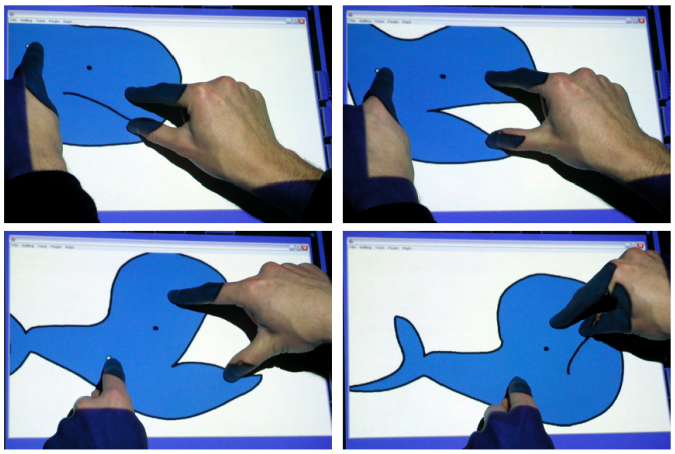

**Figure 36:** *Bimanual manipulation of a drawing.*

## 3.4 Special Purpose Editors

Sketching methods rely on domain knowledge to infer missing information and therefore can be very useful tools designed for specific targets. This section introduces some of those special purpose tools.

### 3.4.1 Tree and Flower Modeling

Okabe et al. [2005] presented a system for quickly and easily designing 3D models of botanical trees using freehand sketches and additional example-based editing operations. The system generates a 3D geometry from a 2D sketch based on the assumption that trees spread their branches so that the distances between the branches are as large as possible. The user can apply additional gesture-based editing operations, such as adding, cutting, and erasing branches. Our system also supports example-based editing modes in which many branches and leaves are generated using a manually designed tree as an example. Studies of user experiences demonstrated that our interface allows novices to design a variety of reasonably natural-looking trees interactively and quickly.

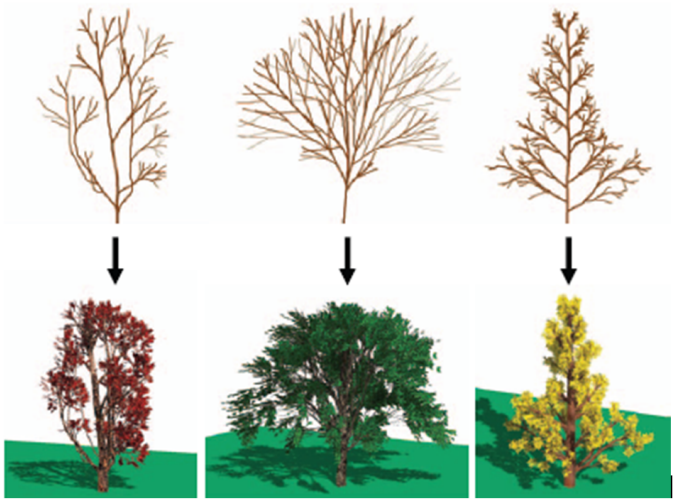

**Figure 37:** *The user sketches a 2D tree and the system automatically generates a 3D tree [Okabe et al. 2005].*

Ijiri et al. [2005] reported a system for modeling flowers in three dimensions quickly and easily while preserving correct botanical structures. We used floral diagrams and inflorescences, which were developed by botanists to concisely describe the structural information of flowers. Floral diagrams represent the layout of floral components on a single flower, while inflorescences are arrangements of multiple flowers. Based on these concepts, we developed a simple user interface that is specially tailored to flower editing, while retaining a maximum variety of generable models. We also provide sketching interfaces to define the geometries of floral components. Separation of structural and geometric editing makes the authoring process more flexible and efficient. We found that even novice users could easily design various flower models using our technique. Our system is an example of application-customized sketching, illustrating the potential power of a sketching interface that is carefully designed for a specific application.

### 3.4.2 Garment Design and Manipulation

Another interesting application area for sketching interface is garment design and manipulation. Any individual character must be dressed in computer graphics applications and garment design is very important. However, as garments are made of flexible materials with very complicated geometry, the typical approach is to design a garment as a collection of rigid thin plates, which are placed around the body, and then run a physical simulation. However, obtaining the desired result is difficult using this approach. Here, we introduce two example systems addressing this problem. One method is to have the user sketch a garment directly on the character and the other is to allow the user

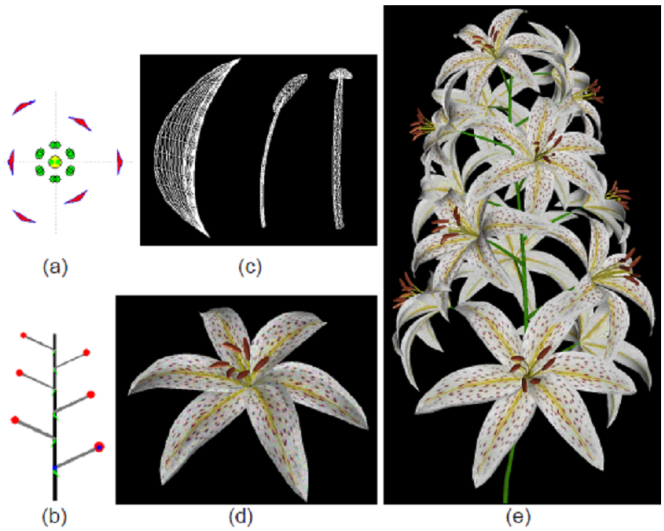


**Figure 38:** *The user defines the layout in the structure editor and the geometry by sketching [Ijiri et al. 2005].*

to place a given garment on the character by sketching.

Turquin et al. [2004] presented a sketching interface for dressing a virtual character. The user sketches the silhouette of the desired garment on the character and the system automatically generates a garment around the character. The system computes the distance field around the body of the 3D character and places the garment such that the distance between the body and garment in the 3D space is uniform across the body surface. The system also allows the user to sketch additional details such as folding.

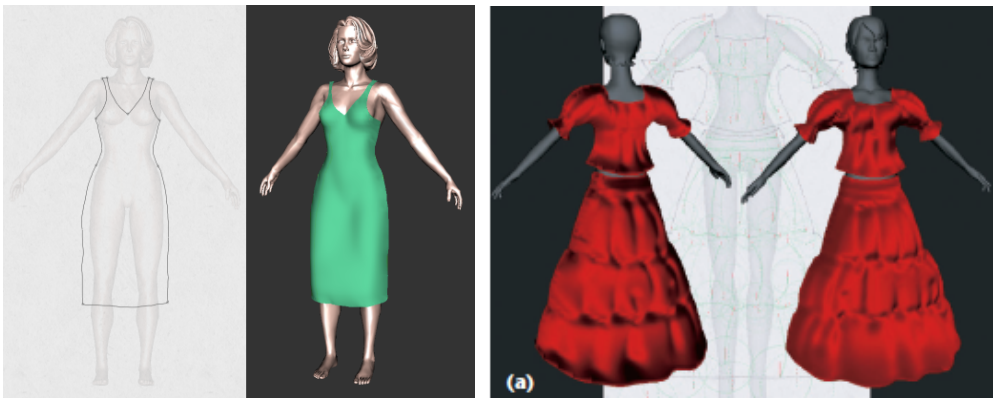


**Figure 39:** *The user draws a 2D silhouette of the desired garment and the system generates a 3D garment [Ijiri et al. 2005].*

The method of Turquin et al. [2004] is mainly for the design of new garments and was not designed specifically to quickly test different ways of putting a garment on a character. Clothing manipulation techniques were developed to address this problem [Igarashi and Hughes 2002]. To put a garment on a character, the user first draws free-form marks on both the garment and the character, indicating positional correspondence (Fig. 40). The system then places the garment on the character so that the marks on the garment match the corresponding marks on the character. The system uses a simple relaxation process during placement to prevent stretching and squashing, even if the lengths of the corresponding marks are different. The user can place reasonably complicated garments on the character with only a few strokes.

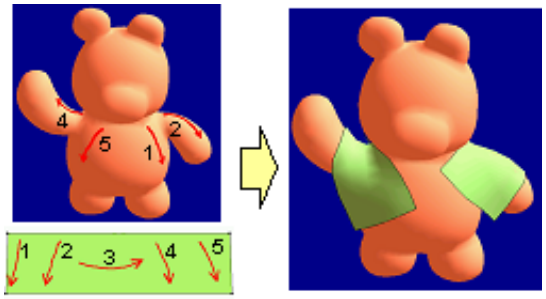


**Figure 40:** *The user draws marks on the character and the garment, and the system places the garment on the character accordingly.*

Once the garment is on the character, the user can grab any point of the garment and drag it onto the surface of the character (Fig. 41). Unlike standard vertex dragging in which a single vertex is moved and relies on subsequent simulation to move the other vertices, this dragging operation directly moves all vertices of the cloth mesh, causing global movement. To achieve this, the movement vector of the dragged vertex is propagated to the complete cloth mesh along the surface of the character.

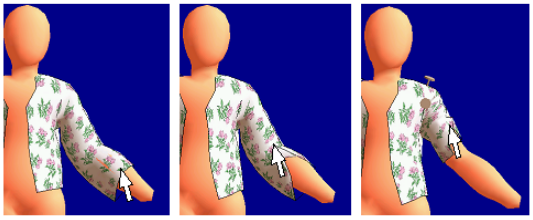

**Figure 41:** *Dragging the garment onto the character. Left and center: traditional approach. Right: our method.*

### 3.5 Summary

This note introduced several sketching systems for 3D modeling and animation design. They begin with the observation of real sketching activities and transfer some of the essence of these activities into the computer. They all infer some missing information in the simple user input using domain knowledge and also provide a disambiguation interface to modify unexpected results. Although specific inference algorithms and interaction techniques introduced here may not be directly applicable to other problems, we believe that the design approaches behind these techniques are general and useful for solving problems in different domains.

# 4 Haptic Interaction with Virtual Environments

For a long time, human beings have dreamed of a virtual world where it is possible to interact with synthetic entities as if they were real. To date, the advances in computer graphics allow us to *see* virtual objects and avatars, to *hear* them, to *move* them, and to *touch* them. It has been shown that the ability to touch virtual objects increases the sense of presence in virtual environments [Insko 2001]. Haptic rendering offers important applicability in engineering and medical training tasks.

This part of the notes focuses on how to empower interactive systems with tactile or haptic feedback. First, we describe the various ways in which the haptic modality can be used in the context of interaction. Then, we focus on one of the interaction paradigms, i.e., haptic rendering, which allows us to perceive contact with a virtual world.

Research in the field of haptics in the last 35 years has covered many more areas than what we have summarised here. [Burdea 1996; McLaughlin et al. 2002; Lin and Otaduy 2008] give a general survey of the field of haptics, as well as a discussion on current research topics.

## 4.1 Introduction

We start by defining some terminology, describing the main interaction paradigms, discussing the evolution of the research in haptic rendering, and introducing practical applications.

### 4.1.1 Definitions

The term *haptic* (from the Greek *haptesthai*, meaning "to touch") is the adjective used to describe something relating to or based on the sense of touch. Haptic is to touching as visual is to seeing and as auditory is to hearing [Fisher et al. 2004].

As described by Klatzky and Lederman [Klatzky and Lederman 2003], touch is one of the main avenues of sensation, and it can be divided into cutaneous, kinesthetic, and haptic systems, based on the underlying neural inputs. The *cutaneous* system employs receptors embedded in the skin, while the *kinesthetic* system employs receptors located in muscles, tendons, and joints. The haptic sensory system employs both cutaneous and kinesthetic receptors, but it differs in the sense that it is associated with an active procedure. Touch becomes active when the sensory inputs are combined with controlled body motion. For example, cutaneous touch becomes active when we explore a surface or grasp an object, while kinesthetic touch becomes active when we manipulate an object and touch other objects with it.

**Figure 42:** *Manipulation of a Virtual Jaw. The user manipulates a haptic device, and the rendering algorithm enables him to perceive forces as if manipulating the upper jaw in the virtual environment.*

*Haptic rendering* is defined as the process of computing and generating forces in response to user interactions with virtual objects [Salisbury et al. 1995]. Several haptic rendering algorithms consider the paradigm of touching virtual objects with a single contact point. Rendering algorithms that follow this description are called 3-DoF (degree-of-freedom) haptic rendering algorithms, because a point in 3D has only three DoFs. Other haptic rendering algorithms deal with the problem of rendering the forces and torques arising from the interaction of two virtual objects. This problem is called 6-DoF haptic rendering, because the grasped object has six DoFs (position and orientation in 3D), and the haptic feedback comprises 3D force and torque. When we eat with a fork, write with a pen, or open a lock with a key, we are moving an object in 3D, and we feel the interaction with other objects. This is, in essence, 6-DoF object manipulation with force-and-torque feedback. Figure 42 shows an example of a user experiencing haptic rendering. When we manipulate an object and touch other objects with it, we perceive cutaneous feedback as the result of grasping, and kinesthetic feedback as the result of contact between objects.

### 4.1.2 Haptic Interaction Paradigms

Haptic human-machine interaction can be perhaps divided into three large categories, based on the purpose of this interaction.

1. **Visualization of data.** In this situation, the original data that is perceived is not necessarily in the tactile modality, but it is transformed into this modality either because it is high-dimensional data, or because collocation with the human motor system may be benefitial. Some examples of haptic visualization of data include the visualization of scientific data (such as fluid dynamics [Lawrence et al. 2000], see Fig. 43), or visuo-haptic segmentation of radiological data [Harders and Szekely 2003] (see Fig. 44).

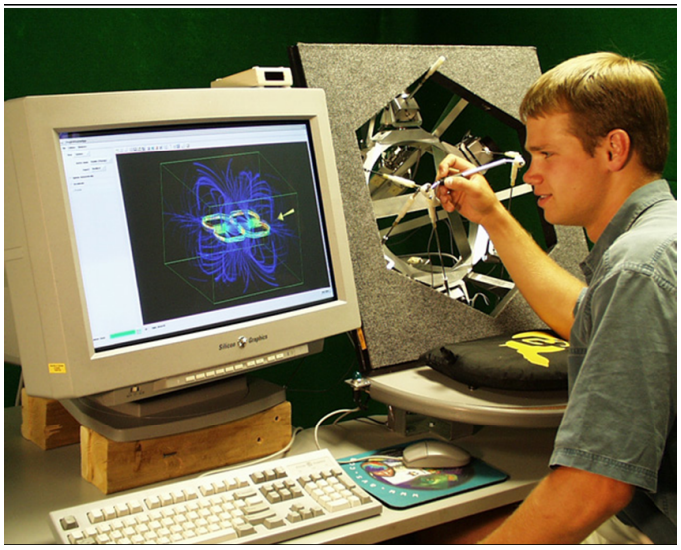

**Figure 43:** *Visual/haptic interface for the display and exploration of fluid-dynamics data.*

2. **Haptic augmentation of cursor actions.** In classical human-computer interaction, the cursor is a merely passive item, but haptics may augment the perception of the user, thus helping discern various signals. Haptic cursor augmentation is classically associated with devices for aiding the visually disabled, but it has recently become of relevance with the explosion of tactile screens (see the TouchScreen concept image from Immersion Corporation in Fig. 45-left) and small-screen hand-held devices [Luk et al. 2006] (see Fig. 45-right).

3. **Tactile interaction with a virtual world.** But, probably, the most common way of haptic interaction is the one that presents a virtual world to the user, and allows the user to move an object in this virtual world and perceive contact with the environment. This is the paradigm best known as haptic rendering, and the rest of this section will focus on it.

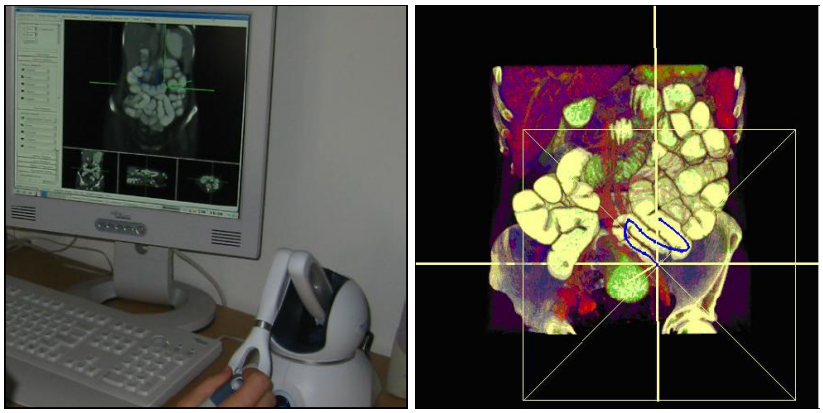

**Figure 44:** *Visuo-haptic segmentation system for the extraction of the small bowel and its centerline.*


**Figure 45:** *Examples of devices for haptic augmentation of interaction.*

### 4.1.3 Kinesthetic Display of Tool Contact

As introduced earlier, haptic perception can be divided into two main categories, based on the nature of the mechanoreceptors that are activated: cutaneous perception is related to mechanoreceptors in the skin, while kinesthetic perception is related to mechanoreceptors in joints, tendons, and muscles. And, the type of contact forces that can be perceived can be classified into two types: forces that appear in direct contact between the skin and the environment, and forces that appear due to contact between an object manipulated by the user (i.e., the *tool*) and other objects in the environment.

Here, we focus mostly on the kinesthetic perception of contact forces through a tool, i.e., the perception of contact between a manipulated tool and other objects, on our joints, tendons, and muscles. Even when using an intermediate tool, subjects can infer medium- and large-scale properties of the objects in the environment as if touching them directly. Moreover, the use of a tool becomes a convenient computational model in the design of haptic rendering algorithms, and even the computation of direct skin interaction could make use of a tool model for representing e.g. fingers.

### 4.1.4 From Telerobotics to Haptic Rendering

In 1965, Ivan Sutherland [Sutherland 1965] proposed a multimodal display that would incorporate haptic feedback into the interaction with virtual worlds. Before that, haptic feedback had already been used mainly in two applications: flight simulators and master-slave robotic teleoperation. The early teleoperator systems had mechanical

linkages between the master and the slave. But, in 1954, Goertz and Thompson [Goertz and Thompson 1954] developed an electrical servomechanism that received feedback signals from sensors mounted on the slave and applied forces to the master, thus producing haptic feedback.

From there, haptic interfaces evolved in multiple directions, but there were two major breakthroughs. The first breakthrough was the idea of substituting the slave robot by a simulated system, in which forces were computed using physically based simulations. The GROPE project at the University of North Carolina at Chapel Hill [Brooks, Jr. et al. 1990], lasting from 1967 to 1990, was the first one to address the synthesis of force feedback from simulated interactions. In particular, the aim of the project was to perform real-time simulation of 3D molecular-docking forces. The second breakthrough was the advent of computer-based Cartesian control for teleoperator systems [Bejczy and Salisbury 1980], enabling a separation of the kinematic configurations of the master and the slave. Later, Cartesian control was applied to the manipulation of simulated slave robots [Kim and Bejczy 1991].

Those first haptic systems were able to simulate the interaction of simple virtual objects only. Perhaps the first project to target computation of forces in the interaction with objects with rich geometric information was Minsky's *Sandpaper* [Minsky et al. 1990]. Minsky et al. developed a planar force feedback system that allowed the exploration of textures. A few years after Minsky's work, Zilles and Salisbury presented an algorithm for 3-DoF haptic rendering of polygonal models [Zilles and Salisbury 1995]. Almost in parallel with Zilles and Salisbury's work, Massie and Salisbury [Massie and Salisbury 1994] designed the PHANTOM, a stylus-based haptic interface that was later commercialised and has become one of the most commonly used force-feedback devices.

But in the late '90s, research in haptic rendering revived one of the problems that first inspired virtual force feedback: 6-DoF haptic rendering or, in other words, grasping of a virtual object and synthesis of kinesthetic feedback of the interaction between this object and its environment.

#### 4.1.5 Haptic Rendering for Virtual Manipulation

Certain professional activities, such as training for high-risk operations or pre-production prototype testing, can benefit greatly from simulated reproductions. The fidelity of the simulated reproductions depends, among other factors, on the similarity of the behaviours of real and virtual objects. In the real world, solid objects cannot interpenetrate. Contact forces can be interpreted mathematically as constraint forces imposed by penetration constraints. However, unless penetration constraints are explicitly imposed, virtual objects are free to penetrate each other in virtual environments. Indeed, one of the most disconcerting experiences in virtual environments is to pass through virtual objects [Insko et al. 2001; Slater and Usoh 1993]. Virtual environments require the simulation of non-penetrating rigid body dynamics, and this problem has been extensively explored in the robotics and computer graphics literature [Baraff 1992; Mirtich 1996].

It has been shown that being able to touch physical replicas of virtual objects (a technique known as *passive haptics* [Insko 2001]) increases the sense of presence in virtual environments. This conclusion can probably be generalised to the case of synthetic cutaneous feedback of the interaction with virtual objects. As reported by Brooks et al. [Brooks, Jr. et al. 1990], kinesthetic feedback radically improved situation awareness in virtual 3D molecular docking. Kinesthetic feedback has proved to enhance task performance in applications such as telerobotic object assembly [Hill and Salisbury 1977], virtual object assembly [Unger et al. 2002], and virtual molecular docking [Ouh-Young 1990]. In particular, task completion time is shorter with kinesthetic feedback in docking operations but not in prepositioning operations.

To summarise, haptic rendering is especially useful in particular examples of training for high-risk operations or pre-production prototype testing activities that involve intensive object manipulation and interaction with the environment. Such examples include minimally invasive or endoscopic surgery [Edmond et al. 1997; Hayward et al. 1998] and virtual prototyping for assembly and maintainability assessment [McNeely et al. 1999; Chen 1999; Andriot 2002; Wan and McNeely 2003] (Fig. 46). Force feedback becomes particularly important and useful in situations with limited visual feedback.

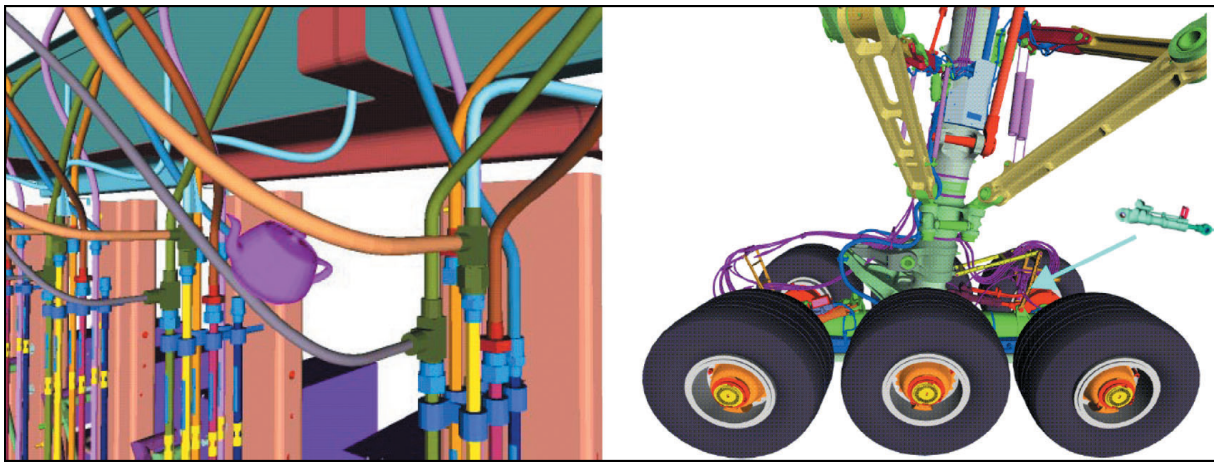

**Figure 46:** *Haptic Rendering Based on Voxelization and Point-Sampling. On the left, scene rendered with the original approach by McNeely et al. [McNeely et al. 1999], and on the right, scene rendered with their improved version [McNeely et al. 2006].*

## 4.2 Stable Interaction Algorithm

A haptic rendering system couples a discrete subsystem (i.e., the rendering algorithm) and a continuous subsystem (i.e., the device and the user). The simulation of a virtual environment using a discrete model, together with sampling and the associated time-delay of the communications, induces a perception of a virtual environment that is not quite an exact replica of the intended continuous model. The practical implication of this difference is that the complete system user-device-algorithm may become unstable when it is perfectly stable in a real (not simulated) world. Much of the work on haptic rendering has targeted the design of stable haptic rendering algorithms.

### 4.2.1 Teleoperation of a Virtual Tool

Figure 42 shows an example of haptic rendering of the interaction between two virtual jaws. The user manipulates a haptic device, and the rendering algorithm allows him to perceive the contact between the jaws as if he were actually holding and moving the upper jaw. In this example, the upper jaw can be regarded as a virtual tool, and the lower jaw constitutes the rest of the virtual environment.

For virtual contact to be perceived as the result of the interaction between a virtual tool and the virtual environment, haptic rendering consists of two key tasks:

1. Compute and display the forces resulting from contact between the virtual tool and the virtual environment.

2. Compute the configuration of the virtual tool.

A haptic rendering system composes of two sub-systems: one real system (i.e., the user and the haptic device), and one virtual system (i.e., the tool and the environment). The tool acts as a virtual counterpart of the haptic device. From this perspective, haptic rendering presents a remarkable similarity to master-slave teleoperation, a topic well studied in the field of robotics. The main difference is that in teleoperation both the master and the slave are real physical systems, while in haptic rendering the tool is a virtual system. Similarly, haptic rendering shares some of the challenges of teleoperation, namely, the computation of *transparent* teleoperation: i.e., the virtual tool should follow the configuration of the device, and the device should produce forces that match those computed on the tool, without filtering or instability artifacts.

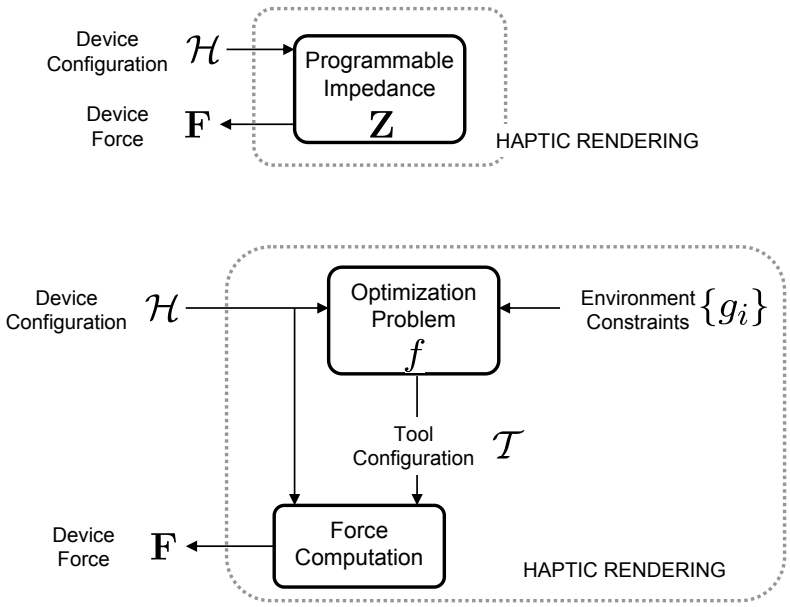


**Figure 47:** *Overview of Haptic Rendering. The top block diagram shows haptic rendering as an impedance control problem, with the device configuration as input, and the device forces as output. A more detailed look in the bottom breaks the haptic rendering problem into two subproblems: (i) computation of the tool configuration, and (ii) computation of device forces.*

### 4.2.2 Tasks of Haptic Rendering

It becomes clear from the discussion above that a haptic rendering problem should address two major computational issues, i.e., finding the tool configuration and computing contact forces, and it should use information about device configuration. Moreover, it requires knowledge about the environment, and in some cases it modifies the environment as well. With these aspects in mind, one possible general definition of the rendering problem could be as follows:

*Given a configuration of the haptic device $\mathcal{H}$, find a configuration of the tool $\mathcal{T}$ that minimizes an objective function $f(\mathcal{H}-\mathcal{T})$, subject to environment constraints. Display to the user a force $\mathbf{F}(\mathcal{H},\mathcal{T})$ dependent on the configurations of the device and the tool.*

This definition assumes a causality precedence where the input variable is the configuration of the haptic device $\mathcal{H}$, and the output variable is the force $\mathbf{F}$. This precedence is known as *impedance rendering*, because the haptic rendering algorithm can be regarded as a programmable mechanical impedance [Hogan 1985; Adams and Hannaford 1998], as depicted in Figure 47. In impedance rendering, the device control system should provide position information and implement a force control loop.

A different possibility is *admittance rendering*, where the haptic rendering system can be regarded as a programmable mechanical admittance that computes the desired device configuration, as the result of input device forces. In that case, the device control should provide device forces and implement a position control loop. As discussed by [Adams and Hannaford 1998], the two types of rendering systems present dual properties and their design can be addressed in a unified manner, therefore here we restrict the exposition to impedance rendering.

In the definition of the haptic rendering problem presented above we have not specified the objective function that must be optimized for computing the tool's configuration nor the function for computing output forces. The dif-

ferences among various haptic rendering algorithms lie precisely in the design of these functions, and are briefly outlined next.

### 4.2.3 Stability and Control Theory

The concept of *mechanical impedance* extends the notion of electrical impedance and refers to the quotient between force and velocity. Hogan [Hogan 1985] introduced the idea of impedance control for contact tasks in manipulation. Earlier techniques controlled contact force, robot velocity, or both, but Hogan suggested controlling directly the mechanical impedance, which governs the dynamic properties of the system. When the end effector of a robot touches a rigid surface, it suffers a drastic change of mechanical impedance, from low impedance in free space, to high impedance during contact. This phenomenon imposes serious difficulties on earlier control techniques, inducing instabilities.

The function of a haptic device is to display the feedback force of a virtual world to a human user. Haptic devices present control challenges very similar to those of manipulators for contact tasks.

A subsystem is *passive* if it does not add energy to the global system. Passivity is a powerful tool for analysing stability of coupled systems, because the coupled system obtained from two passive subsystems is always stable. Colgate and his colleagues were the first to apply passivity criteria to the analysis of stability in haptic rendering of virtual walls [Colgate et al. 1993]. Passivity-based analysis has enabled separate study of the behaviour of the human subsystem, the haptic device, and the virtual environment in force-feedback systems.

Hogan discovered that the human neuromuscular system exhibits externally simple, springlike behaviour [Hogan 1986]. This finding implies that the human arm holding a haptic device can be regarded as a passive subsystem, and the stability analysis can focus on the haptic device and the virtual environment.

Note that human limbs are not passive in all conditions, but the bandwidth at which a subject can perform active motions is very low compared to the frequencies at which stability problems may arise. Some authors [Shimoga 1992; Burdea 1996] report that the bandwidth at which humans can perform controlled actions with the hand or fingers is between $5$ and $10$Hz. On the other hand, sensing bandwidth can be as high as $20$ to $30$Hz for proprioception, $400$Hz for tactile sensing, and $5$ to $10$kHz for roughness perception.

Colgate and Schenkel [Colgate and Schenkel 1994] observed that the oscillations perceived by a haptic user during system instability are a result of active behaviour of the force-feedback system. This active behaviour is a consequence of time delay and loss of information inherent in sampled-data systems, as suggested by others before [Brooks, Jr. et al. 1990]. Colgate and Schenkel formulated passivity conditions in haptic rendering of a virtual wall. For that analysis, they modelled the virtual wall as a viscoelastic unilateral constraint, and they accounted for the continuous dynamics of the haptic device, sampling of the position signal, discrete differentiation for obtaining velocity, and a zero-order hold of the output force. They reached a sufficient condition for passivity that relates the stiffness $K$ and damping $B$ of the virtual wall, the inherent damping $b$ of the device, and the sampling period $T$:

$$b > \frac{KT}{2} + B. \tag{17}$$

After deriving stability conditions for rendering virtual walls modelled as unilateral linear constraints, Colgate and his colleagues considered more complex environments [Colgate et al. 1995]. A general virtual environment is non-linear, and it presents multiple and variable constraints. Their approach enforces a discrete-time passive implementation of the virtual environment and sets a multidimensional viscoelastic *virtual coupling* between the virtual environment and the haptic display. In this way, the stability of the system is guaranteed as long as the virtual coupling is itself passive, and this condition can be analysed using the same techniques as those used for virtual walls [Colgate and Schenkel 1994]. As a result of Colgate's virtual coupling [Colgate et al. 1995], the complexity of the problem was shifted towards designing a passive solution of virtual world dynamics. As noted by Colgate et

al. [Colgate et al. 1995], one possible way to enforce passivity in rigid body dynamics simulation is to use implicit integration with penalty methods.

Adams and Hannaford [Adams and Hannaford 1998] derived stability conditions for coupled systems based on network theory. They also extended the concept of virtual coupling to admittance-type devices. Miller et al. [Miller et al. 1999] extended Colgate's passivity analysis techniques, relaxing the requirement of passive virtual environments but enforcing *cyclo-passivity* of the complete system. Hannaford and his colleagues [Hannaford et al. 2002] investigated the use of adaptive controllers instead of the traditional fixed-value virtual couplings. They designed passivity observers and passivity controllers for dissipating the excess of energy generated by the virtual environment.

### 4.2.4 The Optimization Problem

The simplest possible option for the objective function is the distance between tool and haptic device, i.e., $f = \|\mathcal{H} - \mathcal{T}\|$. This means that the haptic rendering algorithm should simply try to place the tool as close as possible to the haptic device. Given this objective function, one can incorporate information about the environment in multiple ways. In the most simplest way, known as *direct rendering* and described in more detail in the next section, the environment is not accounted for in the optimization, leading to the solution $\mathcal{T} = \mathcal{H}$. Another possibility is to introduce hard inequality constraints $g_i(\mathcal{T}) \geq 0$ that model non-penetration of the tool in the environment. The highly popular *god-object* and *virtual proxy* (Fig. 48) 3-DoF haptic rendering algorithms [Zilles and Salisbury 1995; Ruspini et al. 1997] formulate such an optimization problem. Non-penetration may also be modeled following the penalty method, with soft constraints that are added to the objective function: $f = \|\mathcal{H} - \mathcal{T}\| + \sum_i k_i g_i^2(\mathcal{T})$.

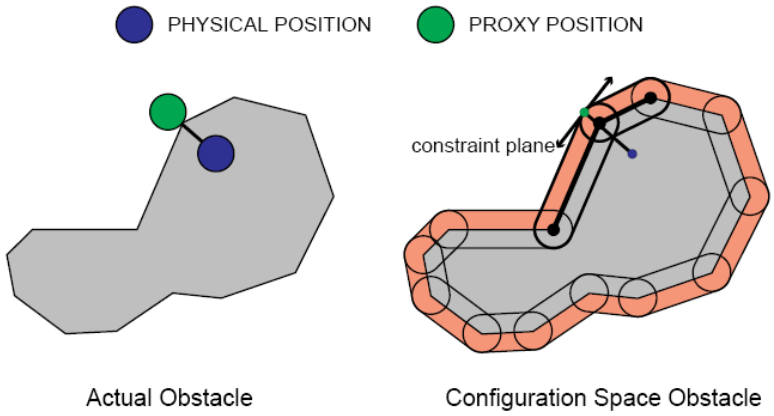


**Figure 48:** *Concept of the virtual proxy 3-DoF rendering algorithm.*

As in general optimization problems, there are multiple ways of solving the tool's configuration. The optimization may be solved until convergence on every rendering frame, but it is also possible to perform a finite number of solver iterations (e.g., simply one), leading to a quasi-static solution for every frame.

Moreover, one could add inertial behavior to the tool, leading to a problem of dynamic simulation. Then, the problem of solving the configuration of the tool can be formulated as a dynamic simulation of a virtual physically-based replica of a real object. This approach has been, in fact, followed by several authors (e.g., [McNeely et al. 1999; Otaduy and Lin 2006]). Modeling the virtual tool as a dynamic object has shown advantages for the analysis of stability of the complete rendering algorithm. If the dynamic simulation can be regarded as a solution to an optimization problem, where forces represent the gradient of the objective function, then the selected type of numerical integration method can be viewed as a different method for solving the optimization problem. For example, a simple explicit Euler corresponds to gradient descent, while implicit Euler corresponds to a Newton solver.

### 4.2.5 Direct Rendering Algorithm

The overall architecture of direct rendering methods is shown in Figure 49. Direct rendering relies on an impedance-type control strategy. First, the configuration of the haptic device is received from the controller, and it is assigned directly to the virtual tool. Collision detection is then performed between the virtual tool and the environment. Collision response is typically computed as a function of object separation or penetration depth using penalty-based methods. Finally, the resulting contact force (and possibly torque) is directly fed back to the device controller.

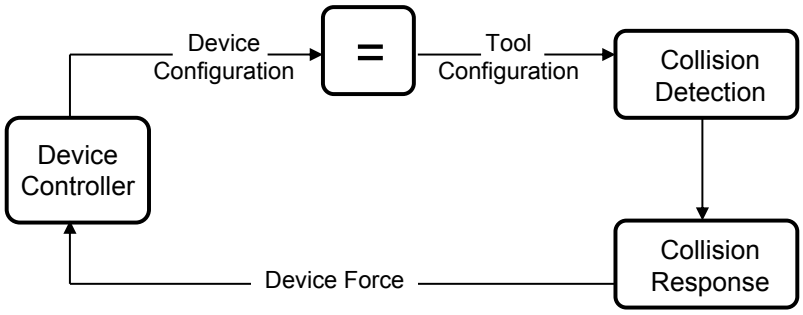


**Figure 49:** *Main Components of a Direct Rendering Algorithm.*

More formally, and following the discussion from Section 4.2.4, direct rendering corresponds to an optimization problem that trivially assigns $\mathcal{T} = \mathcal{H}$ (up to some scale or rigid transformation). This solution answers the second problem in haptic rendering, i.e., the computation of the configuration of the tool. Then, the first problem, the computation of forces, is formulated as a function of the location of the tool in the virtual environment, $\mathbf{F}(g, \mathcal{T})$.

The popularity of direct rendering stems obviously from the simplicity of the calculation of the tool's configuration, as there is no need to formulate a complex optimization problem (for example, rigid body dynamics in 6-DoF rendering). However, the use of penalty methods for force computation has its drawbacks, as penetration values may be quite large and visually perceptible, and system instability can arise if the force update rate drops below the range of stable values.

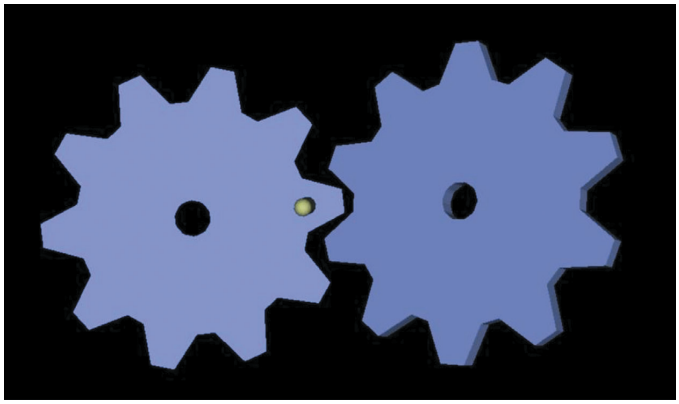

**Figure 50:** *Example of direct 6-DoF haptic rendering [Gregory et al. 2000].*

Throughout the years, direct rendering architectures have often been used as a first practical approach to haptic rendering. Thus, the first 3-DoF haptic rendering algorithms computed forces based on potential fields defined inside the environment. However, as pointed out early by [Zilles and Salisbury 1995], this approach may lead to force discontinuities and pop-through problems. Direct rendering algorithms are perhaps more popular for 6-DoF rendering, and a number of authors have used them, in combination with convex decomposition and hierarchical

collision detection [Gregory et al. 2000; Ehmann and Lin 2001] (See Fig. 50); with parametric surface representations [Nelson et al. 1999]; collision detection hierarchies based on normal cones [Johnson and Cohen 2001; Johnson and Willemsen 2003; Johnson and Willemsen 2004; Johnson et al. 2005] (Fig. 52); or together with fast penetration depth computation algorithms and contact clustering [Kim et al. 2002; Kim et al. 2003] (Fig. 51). In most of these approaches, the emphasis was on fast collision detection or proximity queries, and the work can also be combined with the virtual coupling algorithms described next.

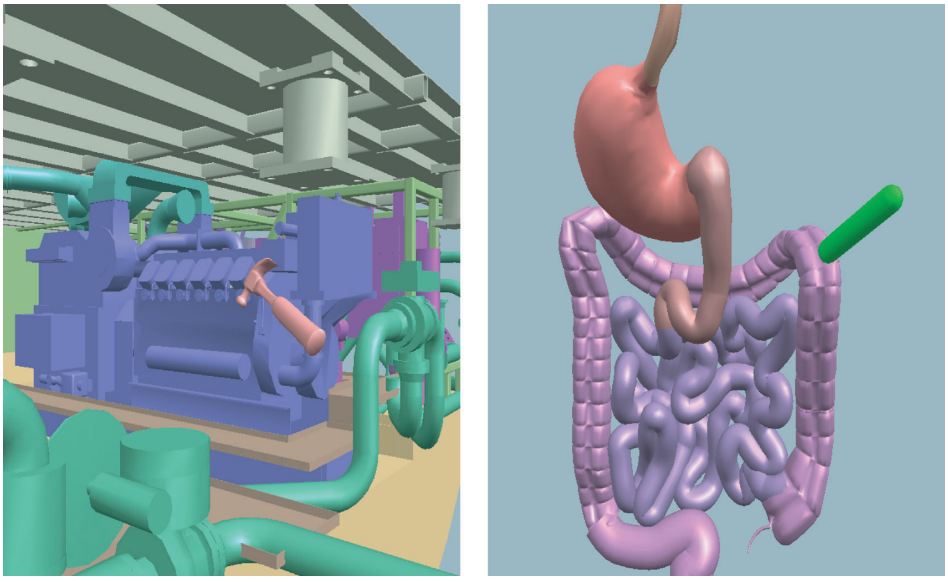

**Figure 51:** *Rendering using fast penetration depth computation [Kim et al. 2003].*

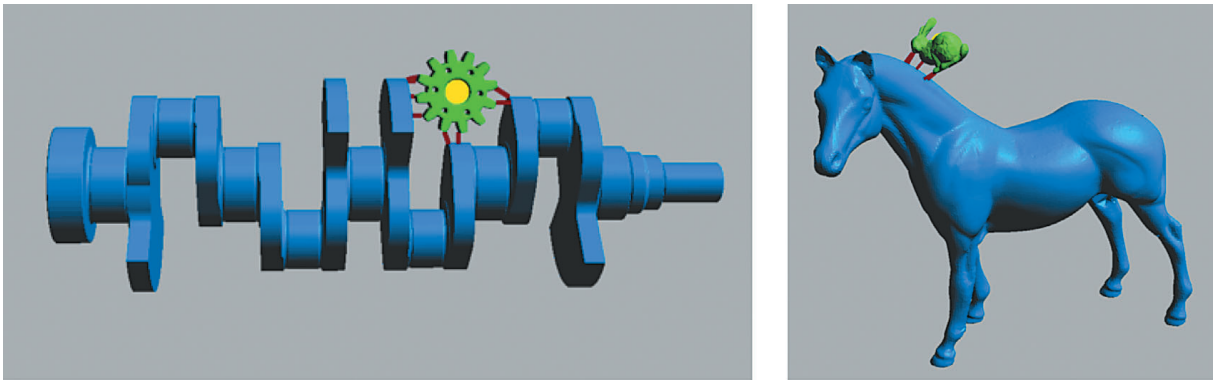

**Figure 52:** *Rendering using hierarchies of normal cones [Kim et al. 2003].*

#### 4.2.6 Rendering through Virtual Coupling

Despite the apparent simplicity of direct rendering, the computation of contact and display forces may become a complex task from the stability point-of-view. Stability of haptic rendering can be answered by studying the range of programmable impedances. With direct rendering and penalty-based contact forces, rendering impedance is hardly controllable, leading often to unstable haptic display.

Stability enforcement can largely be simplified by separating the device and tool configurations, and inserting in-between a viscoelastic link referred to as *virtual coupling* [Colgate et al. 1995]. The connection of passive subsystems through virtual coupling leads to an overall stable system. Figure 53 depicts the configurations of the device

and the tool in a 6-DoF virtual contact scenario using virtual coupling. Contact force and torque are transmitted to the user as a function of the translational and rotational misalignment between tool and device configurations.

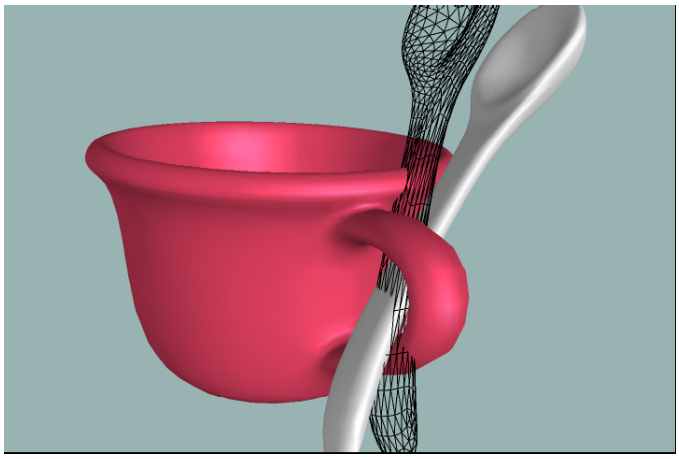

**Figure 53:** *Manipulation through Virtual Coupling. As the spoon is constrained inside the handle of the cup, the contact force and torque are transmitted through a virtual coupling. A wireframe image of the spoon represents the actual configuration of the haptic device [Otaduy and Lin 2006].*

Figure 54 depicts the general structure of a rendering algorithm based on virtual coupling (for an impedance-type display). The input to the rendering algorithm is the device configuration, but the tool configuration is solved in general through an optimization problem, which also accounts for environment constraints. The difference between device and tool configuration is used both for the optimization problem and for computing output device forces.

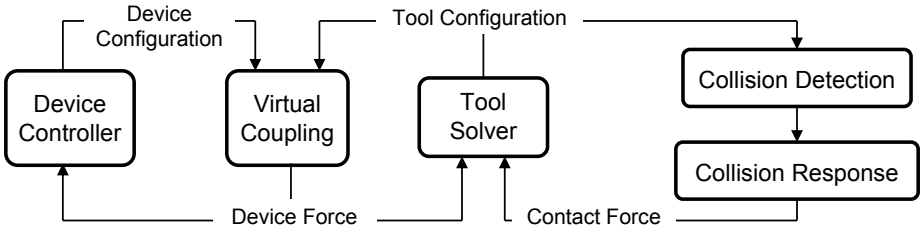


**Figure 54:** *Main Components of a Rendering Algorithm Using Virtual Coupling.*

The most common form of virtual coupling is a viscoelastic spring-damper link. Such a virtual coupling was used by [Zilles and Salisbury 1995; Ruspini et al. 1997] in the god-object and virtual proxy algorithms for 3-DoF rendering. The concept was later extended to 6-DoF rendering [McNeely et al. 1999], by considering translational and rotational springs. For simplicity, here we also group under the name of virtual coupling other approaches that separate tool and device configurations, such as the four-channel architecture based on teleoperation control designed by [Sirouspour et al. 2000], or constraint-aware projections of virtual coupling for 6-DoF rendering [Ortega et al. 2006] (see Fig. 56).

The use of a virtual coupling allows a separate design of the impedance displayed to the user (subject to stability criteria), from the impedance (i.e., stiffness) of environment constraints acting on the tool. Environment constraints can be of high stiffness, which reduces (or even completely eliminates) visible interpenetration problems.

On the other hand, virtual coupling algorithms may suffer from noticeable and undesirable filtering effects, in case the update rate of the haptic rendering algorithm becomes too low, which highly limits the value of the rendering impedance. Multirate algorithms [Adachi et al. 1995; Mark et al. 1996a; Otaduy and Lin 2006] (discussed in more detail in section 4.5) can largely increase the transparency of the rendering by allowing stiffer impedances.

## 4.3 Components of a Rendering Algorithm

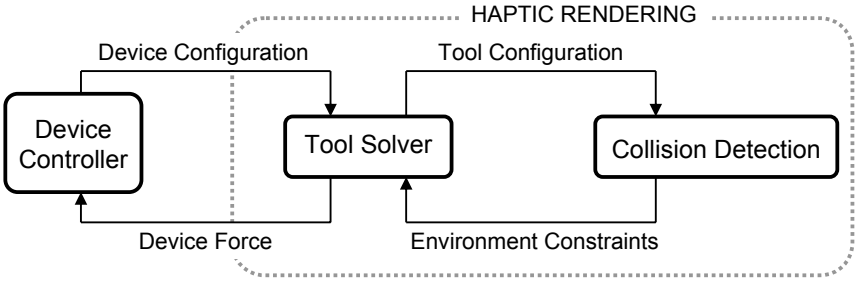


**Figure 55:** *Main Components of a General Impedance-Type Rendering Algorithm.*

On a high level, for an impedance-rendering system, the rendering algorithm must be composed of (i) a solver module that determines the configuration of the tool, and (ii) a collision detection module that defines environment constraints on the tool. Figure 55 depicts such high-level description of the algorithm.

More specifically, we distinguish the following components of the rendering algorithm, which may vary among specific implementations:

### 4.3.1 The Tool Model

The number of degrees of freedom (DoFs) of the tool is a variable that to a large extent depends on the application, and should be minimized as much as possible to reduce the complexity of the rendering. Hence, in many modeling applications, it suffices to describe the tool as a point with three DoFs (translation in 3D), in medical applications, it is often possible to describe it as a ray segment with five DoFs, and in virtual prototyping, it is often required to describe the tool as a solid with six DoFs (translation and rotation in 3D). When the tool is represented as a point, the haptic rendering problem is known as *three-degree-of-freedom* (3-DoF) haptic rendering, and when the tool is represented as a rigid body, it is known as *six-degree-of-freedom* (6-DoF) haptic rendering.

In case of modeling the tool and/or the environment with solid objects, the haptic rendering algorithm also depends on the material properties of these objects. Rigid solids (see section 4.4.1) are limited to six DoFs, while deformable solids (see section 4.4.2) may present a large number of DoFs. At present, the dynamic simulation of a rigid tool is efficiently handled in commodity processors, and the challenge may lie on the complexity of the environment and the contact configuration between tool and environment. Efficient dynamic simulation of complex deformable objects at haptic rates is, however, an issue that deserves further research.

### 4.3.2 Collision Detection

As noted in Figure 55, in the context of haptic rendering, collision detection is the process that, given a configuration of the tool, detects potentially violated environment constraints. Collision detection can easily become the computational bottleneck of a haptic rendering system with geometrically complex objects, and its cost often depends on the configuration space of the contacts. Therefore, we devote much attention to the detection of collisions with a point tool for 3-Dof rendering and the detection of collisions with a solid tool for 6-DoF rendering.

There are also many variations of collision detection depending on the geometric representation of the objects, e.g., polygonal representations, parametric surfaces, or objects with high-resolution textures.

### 4.3.3 Collision Response

In algorithms where the tool's configuration is computed through a dynamic simulation, *collision response* takes the environment constraints given by the collision detection module as input, and computes forces acting on the tool. Therefore, collision response is tightly related to the formulation of environment constraints $g_i(\mathcal{T})$ discussed above.

The two commonly used approaches for collision response are penalty methods and constraint-based methods (e.g., the one by [Ortega et al. 2006], see Fig. 56), which we introduce later in Section 4.4.3. In case of dynamic environments, collision response must be computed on the environment objects as well.

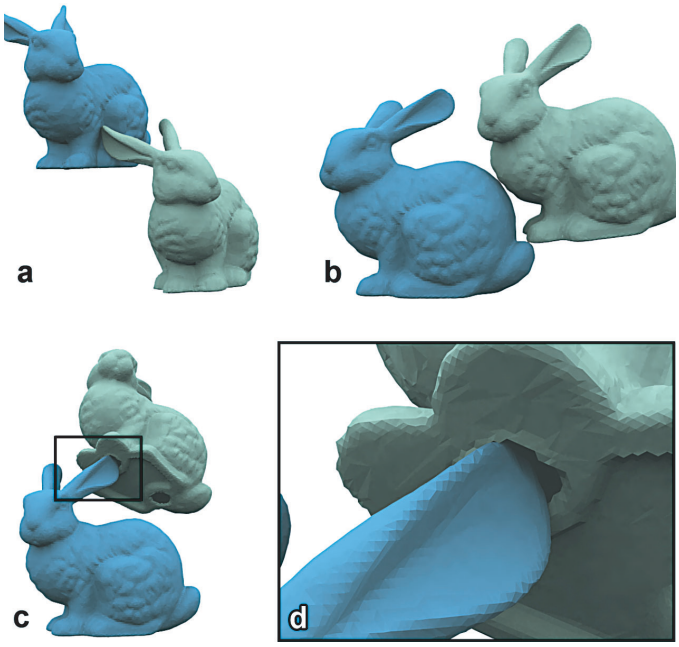


**Figure 56:** *Haptic rendering using the 6-DoF god-object method [Ortega et al. 2006].*

## 4.4 Modeling the Tool and the Environment

In this section we pay special attention to the optimization problem for computing the tool configuration, by briefly introducing some examples: a 6-DoF tool solved with rigid body dynamic simulation, deformable objects, and formulation of contact constraints.

### 4.4.1 Rigid Body Dynamics

We first consider a tool modeled as a rigid body in 3D, which yields 6 DoFs: 3D translation and rotation. One possibility for solving the configuration of the tool is to consider a dynamic model where the tool is influenced by the environment through contact constraint forces, and by the user through virtual coupling force and torque.

We define the generalized coordinates of the tool as $\mathbf{q}$, composed of the position of the center of mass $\mathbf{x}$ and a quaternion describing the orientation $\theta$. We define the velocity vector $\mathbf{v}$ by the velocity of the center of mass $\mathbf{v_x}$ and the angular velocity $\omega$ expressed in the world frame. We denote by $\mathbf{F}$ the generalized forces acting on the tool (i.e., force $\mathbf{F_x}$ and torque $\mathbf{T}$, including gravity, centripetal and Coriolis torque, the action of the virtual coupling, and contact forces).

Given the mass $m$ and the inertia tensor $\mathbf{M}$ of the tool, the dynamics are defined by the Newton-Euler equations:

$$\begin{aligned} m\dot{\mathbf{v}}_{\mathbf{x}} &= \mathbf{F}_{\mathbf{x}}, \\ \mathbf{M}\dot{\omega} &= \mathbf{T} + (\mathbf{M}\omega) \times \omega. \end{aligned} \tag{18}$$

And the relationship between the generalized coordinates and the velocity vector is:

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{v}_{\mathbf{x}}, \\ \dot{\theta} &= \mathbf{G}\omega. \end{aligned} \tag{19}$$

The matrix $\mathbf{G}$ relates the derivative of the quaternion to the angular velocity, and its definition may be found in e.g. [Shabana 1989]. The same reference will serve for an introduction to rigid body dynamics and the derivation of the Newton-Euler equations.

For compactness, it is useful to write the equations of motion in general form:

$$\begin{aligned} \mathbf{M}\dot{\mathbf{v}} &= \mathbf{F}, \\ \dot{\mathbf{q}} &= \mathbf{G}\mathbf{v}. \end{aligned} \tag{20}$$

#### Time Discretization with Implicit Integration

Here, we consider time discretization schemes that yield a linear update of velocities and positions of the form:

$$\begin{aligned} \tilde{\mathbf{M}}\mathbf{v}(i+1) &= \Delta t\tilde{\mathbf{F}}, \\ \mathbf{q}(i+1) &= \Delta t\tilde{\mathbf{G}}\mathbf{v}(i+1) + \mathbf{q}(i). \end{aligned} \tag{21}$$

Note that the updated coordinates $\mathbf{q}(i+1)$ need to be projected afterwards onto the space of valid rotations (i.e., unit quaternions).

One example of linear update is obtained by using Backward Euler discretization with first-order approximation of derivatives. As pointed out by [Colgate et al. 1995], implicit integration of the differential equations describing the virtual environment can ease the design of a stable haptic display. This observation has lead to the design of 6-DoF haptic rendering algorithms with implicit integration of the rigid body dynamics of the tool [Otaduy and Lin 2005; Otaduy and Gross 2007]. Implicit integration also enhances display transparency by enabling stable simulation of the tool with small mass values.

Linearized Backward Euler discretization takes a general ODE of the form $\dot{\mathbf{y}} = \mathbf{f}(\mathbf{y}, t)$ and applies a discretization $\mathbf{y}(i+1) = \mathbf{y}(i) + \Delta t\mathbf{f}(\mathbf{y}(i+1), t(i+1))$, with a linear approximation of the derivatives $\mathbf{f}(\mathbf{y}(i+1), t(i+1)) \approx \mathbf{f}(\mathbf{y}(i), t) + \frac{\partial \mathbf{f}}{\partial \mathbf{y}}(\mathbf{y}(i+1) - \mathbf{y}(i)) + \frac{\partial \mathbf{f}}{\partial t}\Delta t$. For time-independent derivatives, which is our case, this yields an update rule: $\mathbf{y}(i+1) = \mathbf{y}(i) + \Delta t(\mathbf{I} - \Delta t\frac{\partial \mathbf{f}}{\partial \mathbf{y}})^{-1}\mathbf{f}(i)$.

Applying the linearized Backward Euler discretization to (20), and discarding the derivative of the inertia tensor, the terms of the update rule (21) correspond to:

$$\begin{aligned} \tilde{\mathbf{M}} &= \mathbf{M} - \Delta t\frac{\partial \mathbf{F}}{\partial \mathbf{v}} - \Delta t^2\frac{\partial \mathbf{F}}{\partial \mathbf{q}}\tilde{\mathbf{G}}, \\ \tilde{\mathbf{F}} &= \mathbf{F}(i) + \left(\frac{1}{\Delta t}\mathbf{M} - \frac{\partial \mathbf{F}}{\partial \mathbf{v}}\right)\mathbf{v}(i). \end{aligned} \tag{22}$$

As can be inferred from the equations, implementation of implicit integration requires the formulation of Jacobians of force equations $\frac{\partial \mathbf{F}}{\partial \mathbf{q}}$ and $\frac{\partial \mathbf{F}}{\partial \mathbf{v}}$. These Jacobians include the term for inertial forces $(\mathbf{M}\omega) \times \omega$, contact forces (see Section 4.4.3), or virtual coupling (see next). [Otaduy and Lin 2006; Otaduy 2004] formulate in detail these Jacobians.

#### Six-DoF Virtual Coupling

We pay special attention here to modeling viscoelastic virtual coupling for 6-DoF haptic rendering. The tool $\mathcal{T}$ will undergo a force and torque that move it toward the configuration $\mathcal{H}$ of the haptic device, expressed in coordinates of the virtual environment. We assume that the tool undergoes no force when the device's configuration in the reference system of the tool corresponds to a position $\mathbf{x}_c$ and orientation $\theta_c$. We refer to this configuration as *coupling configuration*. Coupling is often engaged at the center of mass of the tool (i.e., $\mathbf{x}_c = 0$), but this is not necessarily true. Coupling at the center of mass has the advantage that coupling force and torque are fully decoupled from each other.

Given configurations $(\mathbf{x}_\mathcal{T}, \theta_\mathcal{T})$ and $(\mathbf{x}_\mathcal{H}, \theta_\mathcal{H})$ for the tool and the device, linear coupling stiffness $k_x$ and damping $b_x$, the coupling force $\mathbf{F}$ on the tool can be defined as:

$$\mathbf{F} = k_x(\mathbf{x}_\mathcal{H} - \mathbf{x}_\mathcal{T} - \mathbf{R}_\mathcal{T}\mathbf{x}_c) + b_x(\mathbf{v}_\mathcal{H} - \mathbf{v}_\mathcal{T} - \omega_\mathcal{T} \times (\mathbf{R}_\mathcal{T}\mathbf{x}_c)). \tag{23}$$

The definition of the coupling torque requires the use of an equivalent axis of rotation $\mathbf{u}$ [McNeely et al. 1999]. This axis of rotation can be defined using the scalar and vector parts ($\Delta\theta_s$ and $\Delta\theta_u$ respectively) of the quaternion $\Delta\theta = \theta_\mathcal{H} \cdot \theta_c^{-1} \cdot \theta_\mathcal{T}^{-1}$ describing the relative coupling orientation between tool and device. Then:

$$\mathbf{u} = 2 \cdot \text{acos}(\Delta\theta_s) \cdot \left( \frac{1}{\sin(\text{acos}(\Delta\theta_s))} \cdot \Delta\theta_u \right). \tag{24}$$

The coupling torque can then be defined using rotational stiffness $k_\theta$ and damping $b_\theta$ as:

$$\mathbf{T} = (\mathbf{R}\mathbf{x}_c) \times \mathbf{F} + k_\theta \mathbf{u} + b_\theta(\omega_\mathcal{H} - \omega_\mathcal{T}). \tag{25}$$

#### Multibody Simulation

Dynamic simulation of the rigid tool itself is not a computationally expensive problem, but the problem becomes considerably more complex if the tool interacts with multiple rigid bodies. In fact, the fast and robust computation of multibody contact is still a subject of research, paying special attention to stacking or friction [Stewart and Trinkle 2000; Mirtich 2000; Milenkovic and Schmidl 2001; Guendelman et al. 2003; Kaufman et al. 2005].

Moreover, with multibody contact and implicit integration it is not possible to write a decoupled update rule (21) for each body. The coupling between bodies may appear (a) in the Jacobians of contact forces when using penalty methods, or (b) through additional constraint equations when using Lagrange multipliers (see Section 4.4.3).

### 4.4.2 Dynamics of Deformable Objects

There are multiple options for modeling deformable objects, and researchers in haptic rendering have often opted for approximate approaches that trade accuracy for computational cost. Here we do not aim at covering in depth the modeling of deformable objects, but rather highlight through an example the inclusion in the complete rendering algorithm. The works of [Müller and Gross 2004; Barbič and James 2007; Duriez et al. 2006; Garre and Otaduy 2009] discuss in more detail practical examples of deformable object modeling for haptic rendering, focusing on fast approximate models and the handling of contact.

The variational formulation of continuum elasticity equations leads to elastic forces defined as the negative gradient of elastic energy $\int_\Omega \sigma \cdot \epsilon \, \mathrm{d}\Omega$, where $\sigma$ and $\epsilon$ represent stress and strain tensors. The various elasticity models differ in the definition of elastic strain or the definition of the relationship between stress and strain. Given the definition of elastic energy, one can reach a discrete set of equations following the finite element method [Zienkiewicz and Taylor 1989]. Typically, the dynamic motion equations of a deformable body may be written as:

$$\mathbf{M} \cdot \dot{\mathbf{v}} = \mathbf{F} - \mathbf{K}(\mathbf{q}) \cdot (\mathbf{q} - \mathbf{q}_0) - \mathbf{D} \cdot \mathbf{v}, \tag{26}$$

$$\dot{\mathbf{q}} = \mathbf{v}. \tag{27}$$

where $\mathbf{M}$, $\mathbf{D}$, and $\mathbf{K}$ represent, respectively, mass, damping, and stiffness matrices. The stiffness matrix captures elastic forces and is, in general, dependent on the current configuration $\mathbf{q}$.

At present times, haptic rendering calls for fast methods for modeling elasticity, and a reasonable approach is to use the linear Cauchy strain tensor, as well as the linear Hookean relationship between stress and strain. Linear strain leads, however, to considerable artifacts under large deformations, which can be eliminated by using corotational methods that measure deformations in the unrotated setting of each mesh element [Müller et al. 2002; Müller and Gross 2004].

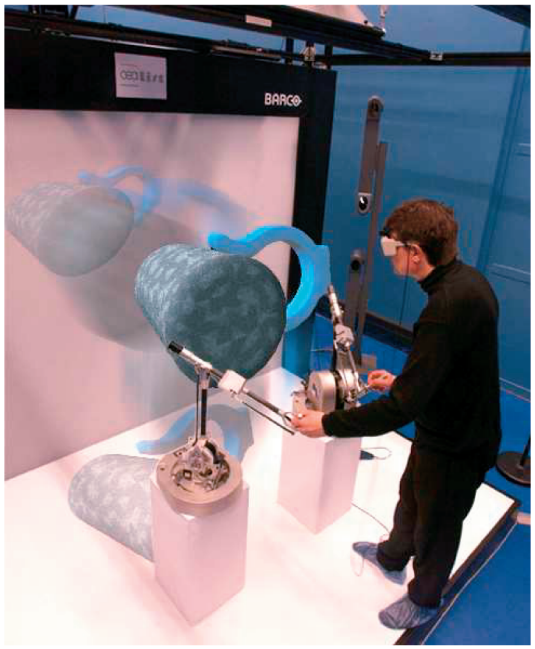


**Figure 57:** *A snapping task involving deformable objects modeled with FE methods [Duriez et al. 2006].*

The use of corotational strain allows for stable and robust implicit integration methods while producing a linear update rule for each time step. With linearized Backward Euler (described before for rigid bodies), the update rule becomes:

$$\tilde{\mathbf{M}}\mathbf{v}(i+1) = \Delta t \tilde{\mathbf{F}}, \tag{28}$$

$$\mathbf{q}(i+1) = \Delta t \mathbf{v}(i+1) + \mathbf{q}(i). \tag{29}$$

The discrete mass matrix $\mathbf{M}$ and force vector $\mathbf{F}$ become:

$$\tilde{\mathbf{M}} = \mathbf{M} + \Delta t \left( \mathbf{D} - \frac{\partial \mathbf{F}}{\partial \mathbf{v}} \right) + \Delta t^2 \left( \mathbf{K}(\mathbf{q}) - \frac{\partial \mathbf{F}}{\partial \mathbf{q}} \right),$$

$$\tilde{\mathbf{F}} = \mathbf{F}(i) + \left( \frac{1}{\Delta t}\mathbf{M} + \mathbf{D} - \frac{\partial \mathbf{F}}{\partial \mathbf{v}} \right) \mathbf{v}(i). \tag{30}$$

References such as [Müller and Gross 2004; Duriez et al. 2006; Garre and Otaduy 2009] offer a more detailed discussion on the efficient simulation of linear elastic models with corotational methods, as well as the implementation of virtual coupling with a deformable tool.

### 4.4.3 Contact Constraints

Contact constraints model the environment as algebraic equations in the configuration space of the tool, $g_i(\mathcal{T}) \geq 0$. A configuration of the tool $\mathcal{T}_0$ such that $g_i(\mathcal{T}_0) = 0$ indicates that the tool is exactly in contact with the environment.

Collision response exerts forces on the tool such that environment constraints are not violated. We will look at two specific ways of modeling environment contact constraints, penalty-based methods and Lagrange multipliers, and we will focus as an example on their application to a rigid tool.

### Penalty Method

In general terms, the penalty method models contact constraints as springs whose elastic energy increases with object interpenetration. Penalty forces are computed as the negative gradient of the elastic energy, which produces collision response that moves objects toward a non-penetrating configuration.

For simplicity, we will consider here linearized point-on-plane contacts. Given a colliding point $\mathbf{p}$ of the tool, a general contact constraint has the form $g_i(\mathbf{p}) \geq 0$, and after linearization $\mathbf{n}^T(\mathbf{p} - \mathbf{p}_0) \geq 0$, where $\mathbf{n} = \nabla g_i$ is the normal of the constraint (i.e., the normal of the contact plane), and $\mathbf{p}_0$ is the contact point on the environment. With such a linearized constraint, penetration depth can easily be defined as $\delta = -\mathbf{n}^T(\mathbf{p} - \mathbf{p}_0)$.

Penalty energies can be defined in multiple ways, but the simplest is to consider a Hookean spring, which yields an energy $E = \frac{1}{2}k\delta^2$, where $k$ is the contact stiffness. Then, the contact penalty force becomes $\mathbf{F} = -\nabla E = -k\delta\nabla\delta$. It is also common to apply penalty forces when objects become closer than a certain tolerance $d$, which can be easily handled by redefining the penetration depth as $\delta = d - \mathbf{n}^T(\mathbf{p} - \mathbf{p}_0)$. The addition of a tolerance has two major advantages: the possibility of using penalty methods in applications that do not allow object interpenetration, and a reduction of the cost of collision detection. With the addition of a tolerance, object interpenetration occurs less frequently, and computation of penetration depth is notably more costly than computation of separation distance.

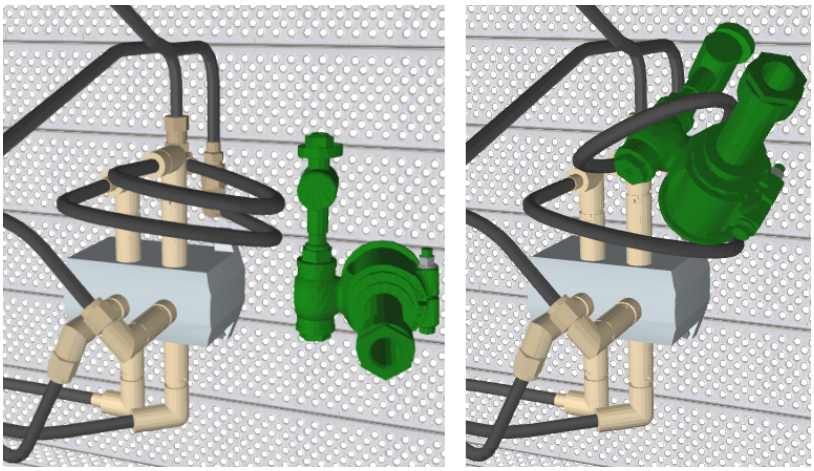

**Figure 58:** *Example of complex scene with deformable bodies rendered using time-critical methods [Barbič and James 2007].*

With a rigid tool, the contact point $\mathbf{p}$ can be expressed in terms of the rigid body state as $\mathbf{p} = \mathbf{x} + \mathbf{R}\mathbf{r}$, where $\mathbf{x}$ and $\mathbf{R}$ are, respectively, the position and orientation of the tool, and $\mathbf{r}$ is the position of the contact point in the tool's reference frame. Then, for the case of a rigid tool, and adding a damping term $b$, the penalty force and torque are:

$$\begin{aligned} \mathbf{F} &= -k\mathbf{N}(\mathbf{x} + \mathbf{R}\mathbf{r} - \mathbf{p}_0) - b\mathbf{N}(\mathbf{v} + \omega \times (\mathbf{R}\mathbf{r})), \\ \mathbf{T} &= (\mathbf{R}\mathbf{r}) \times \mathbf{F}. \end{aligned} \tag{31}$$

$\mathbf{N} = \mathbf{n}\mathbf{n}^T$ is a matrix that projects a vector onto the normal of the constraint plane.

Penalty-based methods offer several attractive properties: the force model is local to each contact and computationally simple, object interpenetration is inherently allowed, and the cost of the numerical integration is almost insensitive to the complexity of the contact configuration. This last property makes penalty-based methods well

suited for interactive applications with fixed time steps, such as haptic rendering. In fact, penalty-based methods have been applied in many 6-DoF haptic rendering approaches [McNeely et al. 1999; Kim et al. 2003; Johnson and Willemsen 2003; McNeely et al. 2006; Otaduy and Lin 2006; Barbič and James 2007].

However, penalty-based methods also have some disadvantages. For example, there is no direct control over physical parameters, such as the coefficient of restitution, and friction forces are difficult to model, as they require tracking contact points and using local methods [Karnopp 1985; Hayward and Armstrong 2000]. But, most importantly, geometric discontinuities in the location of contact points and/or normals leads to torque discontinuities, as depicted schematically in Figure 59. Different authors have proposed various definitions for contact points and normals, with various advantages and drawbacks. [McNeely et al. 1999; McNeely et al. 2006; Barbič and James 2007] sample the objects with a discrete set of points, and define contact points as the penetrating subset. [Johnson and Willemsen 2003; Otaduy and Lin 2006], on the other hand, employ continuous surface definitions, and define contact points as local extrema of the distance function between colliding surfaces. Using a fixed discrete set of points allows for increased force continuity, while using continuous surface definitions allows for the detection of all interpenetrations. With the strict definition of penalty energy given above, penalty force normals are defined as the gradient of penetration depth, which is discontinuous on the medial axis of the objects. [McNeely et al. 1999; McNeely et al. 2006; Barbič and James 2007] avoid this problem by defining as contact normal the surface normal at each penetrating point. This alternative definition is continuous in time, but does not guarantee that contact forces reduce interpenetration.

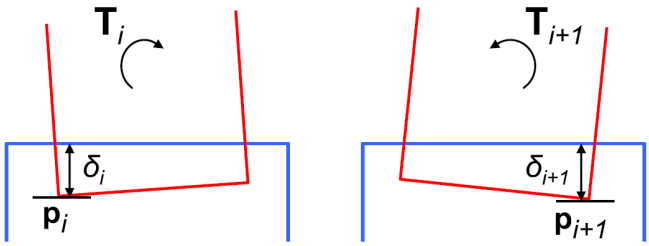


**Figure 59:** *Torque Discontinuity: (a) penetration depth and torque at time $t_i$, with contact point $\mathbf{p}_i$; (b) penetration depth and torque at time $t_{i+1}$, after the contact moves to contact point $\mathbf{p}_{i+1}$.*

With penalty-based methods, non-penetration constraints are enforced by means of very high contact stiffness, which could yield instability problems if numerical integration is executed using fast explicit methods. The use of implicit integration of the tool, as described in Section 4.4.1, enhances stability in the presence of high contact stiffness [Wu 2000; Larsen 2001; Otaduy and Lin 2006; Barbič and James 2007]. However, the dynamic equations of the different dynamic bodies (see (21) for rigid bodies or (29) for deformable bodies) become then coupled, and a linear system must be solved for each contact group. We refer to [Otaduy and Lin 2006] for further details on the Jacobians of penalty force and torque for 6-DoF haptic rendering.

#### Lagrange Multipliers

The method of Lagrange multipliers allows for an exact enforcement of contact constraints $\mathbf{g}(\mathcal{T}) \geq 0$ by modeling workless constraint forces $\mathbf{F}_c = \mathbf{J}^T\lambda$ normal to the constraints. Here we consider multiple constraints grouped in a vector $\mathbf{g}$, and their generalized normals are gathered in a matrix $\mathbf{J}^T = \nabla\mathbf{g}$. Constraint forces are added to regular forces of the dynamic equations of a colliding object (e.g., the tool). Then, constraints and dynamics are formulated in a joint differential algebraic system of equations. The "amount" of constraint force $\lambda$ is the unknown of the system, and it is solved such that constraints are enforced.

Typically, contact constraints are nonlinear, but the solution of constrained dynamics systems can be accelerated by linearizing the constraints. Given the state $\mathbf{q}(i)$ of the tool at a certain time, constraint linearization yields $\mathbf{g}(i+1) \approx \mathbf{g}(i) + \Delta t\mathbf{J} \cdot \mathbf{v}(i+1)$. This linearization, together with the discretized state update equation yields the

following system to be solved per simulation frame:

$$\tilde{\mathbf{M}} \cdot \mathbf{v}(i+1) = \Delta t \tilde{\mathbf{F}} + \mathbf{J}^T \lambda,$$
$$\mathbf{J} \cdot \mathbf{v}(i+1) \geq -\frac{1}{\Delta t}\mathbf{g}(i). \tag{32}$$

The addition of constraints for non-sticking forces $\lambda \geq 0,\ \lambda^T \mathbf{g}(\mathbf{q}) = 0$ yields a linear complementarity problem (LCP) [Cottle et al. 1992], which combines linear equalities and inequalities. The problem in equation (32) is a mixed LCP, and can be transformed into a strict LCP through algebraic manipulation:

$$\mathbf{J}\tilde{\mathbf{M}}^{-1}\mathbf{J}^T\lambda \geq -\frac{1}{\Delta t}\mathbf{g}(i) - \Delta t \mathbf{J}\tilde{\mathbf{M}}^{-1}\tilde{\mathbf{F}}. \tag{33}$$

The LCP can be solved through various techniques [Cottle et al. 1992], and once the Lagrange multipliers $\lambda$ are known, it is possible to update the state of the tool.

There are other variants of the problem, for example by allowing sticking forces through equality constraints, or differentiating the constraints and expressing them on velocities or accelerations. Several of these variants of contact constraints with Lagrange multipliers have been employed in practical solutions to haptic rendering. The god-object method of [Zilles and Salisbury 1995] can be considered as the first application of Lagrange multipliers for contact in 3-DoF haptic rendering. The work of [Ortega et al. 2006] describes an extension of the god-object method to 6-DoF rendering, and [Duriez et al. 2006] formulates in detail frictional contact for haptic rendering of deformable objects. [Garre and Otaduy 2009] propose haptic rendering between deformable objects using an intermediate representation and the constrained dynamics solver from [Otaduy et al. 2009].

Constraint-based methods with Lagrange multipliers handle all concurrent contacts in a single computational problem and attempt to find contact forces that produce physically and geometrically valid motions. As opposed to penalty-based methods, they solve one global problem, which allows, for example, for relatively easy inclusion of accurate friction models. However, constraint-based are computationally expensive, even for the linearized system (32), and the solution of constrained dynamics and the definition of constraints (i.e., collision detection) are highly intertwined. The full problem of constrained dynamics is highly nonlinear, but there are various time-stepping approaches that separate a collision-free dynamics update, collision detection, and collision response, for solving locally linear problems [Bridson et al. 2002; Cirak and West 2005]. Fast enforcement of constrained motion is, however, still a topic of research in haptic rendering, in particular for rendering deformable objects.

## 4.5 Multirate Algorithm

As discussed in Section 4.2.6, rendering algorithms based on virtual coupling [Colgate et al. 1995] can serve to easily design stable rendering. However, the complexity of tool and environment simulation may require low update rates, which turn into low admissible coupling stiffness, and hence low-quality rendering.

Independently of the simulation and collision detection methods employed, and the mechanical characteristics of the tool or the environment, a common solution for enhancing the quality and transparency of haptic rendering is to devise a multirate algorithm (See [Barbagli et al. 2003] for stability analysis of multirate algorithms). A slow process computes accurate interaction between the tool and the environment, and updates an approximate but simple intermediate representation [Adachi et al. 1995]. Then, a fast process synthesizes the forces to be sent to the device, using the intermediate representation. There have been two main approaches for designing intermediate representations, which we discuss next.

### 4.5.1 Geometric Vs. Algebraic Intermediate Representations

One approach is to design a local and/or coarse geometric representation of the tool and/or the environment. A slow thread performs a computation of the interaction between the full representations of the tool and the environment,

and updates the local coarse representation. In parallel, a fast thread computes the interaction between tool and environment using the simplified representations. The fast computation involves identifying simplified contact constraints, through collision detection, and computing the rendering forces. Note that this approach can be used in the context of both virtual coupling algorithms or direct rendering.

The earliest example of multirate rendering by [Adachi et al. 1995] computes collision detection between a point tool and the environment in the slow thread, approximates the environment as a plane, and then uses the plane representation in the fast thread. A similar approach was followed by [Mark et al. 1996b], with addition of plane filtering between local model updates. Others used meshes of different resolutions coupled through Norton equivalents [Astley and Hayward 1998], or local linearized submeshes for approximating high-frequency behavior [Çavuşoğlu and Tendick 2000]. Recently, [Johnson et al. 2005] have suggested the use of local collision detection algorithms for updating the contact constraints in the fast loop.

The second approach is to design a simplified representation of the collision response model between the tool and the environment. The slow thread performs full computation of collision detection and response between tool and environment, and updates a simplified version of the collision response model. This model is then used in the fast thread for computing collision response for rendering forces to the user. The main difference with the geometric approach is that the fast thread does not recompute collision detection for defining contact constraints.

Early approaches to multirate simulation of deformable models considered force extrapolation for defining the local algebraic model [Picinbono et al. 2000]. There are other recent approaches that identify contact constraints in the slow thread, and then use those constraints for force computation in the fast thread, for rigid bodies [Ortega et al. 2006], or for deformable bodies [Duriez et al. 2004]. Apart from those, we should mention the use of contact constraints for computing a least-squares solution to Poisson's restitution hypothesis for rigid bodies [Constantinescu et al. 2005]. Last, the rest of this section describes two examples that compute in the slow thread a linear model of the contact response between the tool and the environment, and then simply evaluate this linear model in the fast thread, for penalty-based methods [Otaduy and Lin 2005; Otaduy and Lin 2006] or for constraint-based methods with Lagrange multipliers [Otaduy and Gross 2007].

### 4.5.2 Example 1: Multirate Rendering with Penalty Methods

Figure 60 shows the structure of the rendering algorithm suggested by [Otaduy and Lin 2005; Otaduy and Lin 2006]. The *visual thread* computes collision detection between the tool and the environment, as well as collision response using the penalty-based method (See section 4.4.3). Moreover, the equations of collision response are linearized, and the linear model is fed to the *haptic thread*. The haptic thread runs at fast haptic update rates, solving for the configuration of the tool subject to forces computed using the linearized contact model. Figure 61 shows one application scenario of the multirate rendering algorithm.

[Otaduy and Lin 2005; Otaduy and Lin 2006] applied the linearization of penalty-based forces to 6-DoF haptic rendering with a rigid tool. Recall equation (31), which describes penalty forces for a rigid tool. Assuming that the visual thread computes collision detection for a configuration $(\mathbf{q}_0, \mathbf{v}_0)$ of the tool, a penalty-based contact model can be linearized in general as:

$$\mathbf{F}_c(\mathbf{q}, \mathbf{v}) \approx \mathbf{F}_c(\mathbf{q}_0, \mathbf{v}_0) + \frac{\partial \mathbf{F}_c}{\partial \mathbf{q}}(\mathbf{q} - \mathbf{q}_0) + \frac{\partial \mathbf{F}_c}{\partial \mathbf{v}}(\mathbf{v} - \mathbf{v}_0). \tag{34}$$

For more details on the linear approximation for a rigid tool, we refer to [Otaduy and Lin 2005; Otaduy and Lin 2006].

An interesting observation of the linearized penalty-based method is that it imposes no additional cost if the rendering algorithm computes dynamics of the tool with implicit integration. As shown in equation (22), the definition of discrete-time inertia requires the same Jacobians $\frac{\partial \mathbf{F}}{\partial \mathbf{q}}$ and $\frac{\partial \mathbf{F}}{\partial \mathbf{v}}$ as the linearized contact model. We would like to point

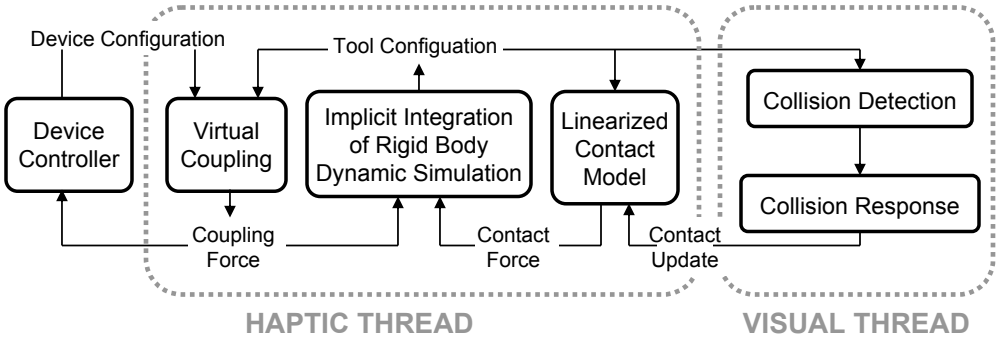


**Figure 60:** *Multirate Rendering Architecture with a Linearized Contact Model. A haptic thread runs at force update rates simulating the dynamics of the tool and computing force feedback, while a visual thread runs asynchronously and updates the linearized contact model [Otaduy and Lin 2005; Otaduy and Lin 2006].*

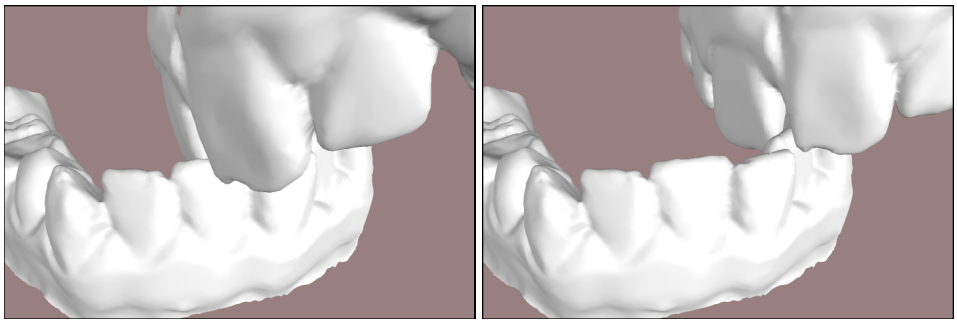

**Figure 61:** *Virtual Interaction Using a Linearized Contact Model. Dexterous interaction of an upper jaw (47,339 triangles) being moved over a lower jaw (40,180 triangles), using the method by [Otaduy and Lin 2005; Otaduy and Lin 2006]. (©2005 IEEE)*

out that these Jacobians are also used in quasi-static methods for solving the configuration of the tool [Wan and McNeely 2003; Barbič and James 2007].

#### 4.5.3 Example 2: Multirate Rendering with Constraints

Figure 62 shows the overall structure of the multirate rendering algorithm presented by [Otaduy and Gross 2007] for 6-DoF haptic rendering between a rigid tool and a deformable environment. This algorithm creates two instances of the rigid tool manipulated by the user. The visual thread, typically running at a low update rate (as low as tens of Hz), performs a full simulation of the *visual tool* coupled to the haptic device and interacting with a deformable environment. The haptic thread, running at a fast update rate of typically 1 kHz, performs the simulation of the *haptic tool* and computes force values to be rendered by the haptic device. Collision detection and full constraint-based collision response are only computed in the visual thread. At the same time, the parameters of a linear contact model are updated, and fed to the haptic thread. This linear model can be evaluated with a fixed, low number of operations, and ensures extremely fast update of contact forces in the haptic thread.

For penalty-based collision response, [Otaduy and Lin 2005] proposed a linearized contact model in the state space of the tool. However, for constraint-based collision response, [Otaduy and Gross 2007] proposed a model of *contact Jacobian*, linearly relating contact forces $\mathbf{F}_c$ and the rest of the forces $\tilde{\mathbf{F}}$ acting on the tool. The linearized model takes the form:

$$\mathbf{F}_c(\tilde{\mathbf{F}}) \approx \mathbf{F}_c(\tilde{\mathbf{F}}_0) + \frac{\partial \mathbf{F}_c}{\partial \tilde{\mathbf{F}}}(\tilde{\mathbf{F}} - \tilde{\mathbf{F}}_0). \tag{35}$$

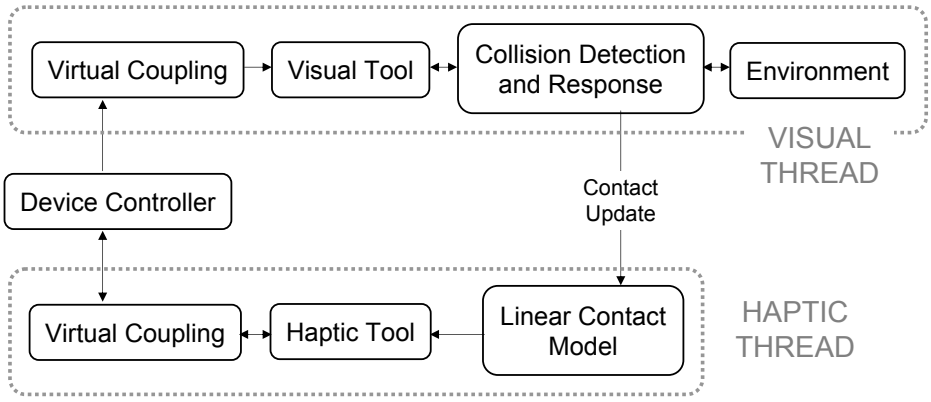


**Figure 62:** *Multirate Rendering Using a Discrete-Time Contact Jacobian [Otaduy and Gross 2007].*

All that needs to be done in the visual thread is to compute the contact Jacobian $\frac{\partial \mathbf{F}_c}{\partial \tilde{\mathbf{F}}}$.

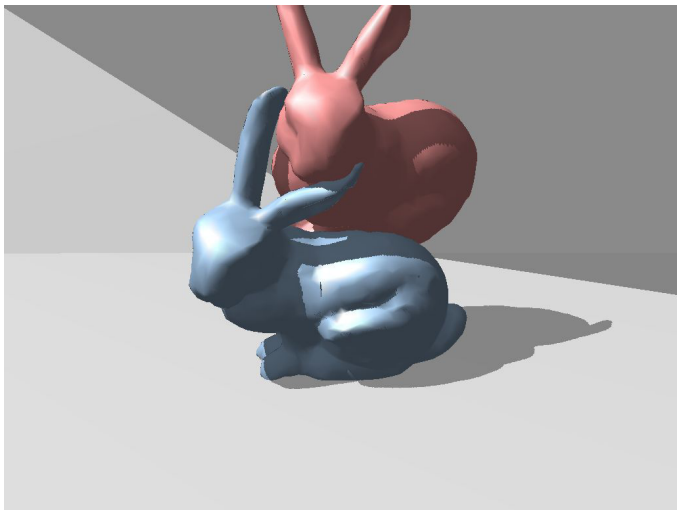

**Figure 63:** *Deformable models rendered using constraint-based contact and intermediate representations [Garre and Otaduy 2009].*

The LCP formulation (33) for collision response can be compactly rewritten as $\mathbf{A}_\lambda \lambda \geq \mathbf{b}_\lambda$. The resolution of the LCP yields a set of inactive contacts, for which the contact force is zero, and a set of active contacts for which the constraints hold exactly $\mathbf{A}_{\lambda,a}\lambda_a = \mathbf{b}_a \Rightarrow \lambda_a = \mathbf{A}_{\lambda,a}^{-1}\mathbf{b}_a$. The contact force can then be written in terms only of active contacts as $\mathbf{F}_c = \mathbf{J}_a^T \mathbf{A}_{\lambda,a}^{-1}\mathbf{b}_a$. Then, the contact Jacobian can be easily formulated as:

$$\frac{\partial \mathbf{F}_c}{\partial \tilde{\mathbf{F}}} = \frac{\partial \mathbf{F}_c}{\partial \mathbf{b}_a} \cdot \frac{\partial \mathbf{b}_a}{\partial \tilde{\mathbf{F}}} = -\Delta t \mathbf{J}_a^T \mathbf{A}_{\lambda,a}^{-1} \mathbf{J}_a \tilde{\mathbf{M}}^{-1}. \tag{36}$$

This formulation involves several approximations, such as ignoring the change of active constraints between time steps or changes of inertia. Note also that equation (36) should be slightly modified to account for a moving or deforming environment, as the state of the tool and the environment are not explicitly separated. Multirate algorithms enable programming very high rendering stiffness, under the assumption that the contact space changes slowly. This is in fact the case in many situations, especially during sliding contact between tool and environment.

The approach of multirate rendering using force linearization has recently been extended to the contact between a deformable tool and other deformable objects in the environment [Garre and Otaduy 2009] (See Fig. 63). The key

of this method is to identify a rigid handle in the deformable tool, which serves as the space for linearization. With a multirate algorithm, the visual thread is simulated using accurate constraint-based contact solvers.

## References


ADACHI, Y., KUMANO, T., AND OGINO, K. 1995. Intermediate representation for stiff virtual objects. *Virtual Reality Annual International Symposium*, 203–210.

ADAMS, R. J., AND HANNAFORD, B. 1998. A two-port framework for the design of unconditionally stable haptic interfaces. *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*.

ANDRIOT, C. 2002. Advances in virtual prototyping. *Clefs CEA Vol. 47, Research and Simulation*.

ASTLEY, O. R., AND HAYWARD, V. 1998. Multirate haptic simulation achieved by coupling finite element meshes through norton equivalents. *Proc. of IEEE International Conference on Robotics and Automation*.

BARAFF, D. 1992. *Dynamic simulation of non-penetrating rigid body simulation*. PhD thesis, Cornell University.

BARBAGLI, F., PRATTICHIZZO, D., AND SALISBURY, K. 2003. Dynamic local models for stable multi-contact haptic interaction with deformable objects. *Proc. of Haptics Symposium*.

BARBIČ, J., AND JAMES, D. L. 2007. Time-critical distributed contact for 6-DoF haptic rendering of adaptively sampled reduced deformable models. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

BEJCZY, A., AND SALISBURY, J. K. 1980. Kinematic coupling between operator and remote manipulator. *Advances in Computer Technology Vol. 1*, 197–211.

BOTT, J., CROWLEY, J., AND LAVIOLA, J. 2009. Exploring 3d gestural interfaces for music creation in video games. In *Proceedings of The Fourth International Conference on the Foundations of Digital Games 2009*, 18–25.

BOWMAN, D. A., AND HODGES, L. F. 1997. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 35–ff.

BOWMAN, D. A., KOLLER, D., AND HODGES, L. F. 1997. Travel in immersive virtual environments: An evaluation of viewpoint motion control techniques. In *Proceedings of the Virtual Reality Annual International Symposium*, 45–52.

BOWMAN, D. A., WINEMAN, J., HODGES, L. F., AND ALLISON, D. 1998. Designing animal habitats within an immersive ve. *IEEE Comput. Graph. Appl. 18*, 5, 9–13.

BOWMAN, D. A., KRUIJFF, E., LAVIOLA, J. J., AND POUPYREV, I. 2004. *3D User Interfaces: Theory and Practice*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. In *Proc. of ACM SIGGRAPH*.

BROOKS, JR., F. P., OUH-YOUNG, M., BATTER, J. J., AND KILPATRICK, P. J. 1990. Project GROPE — Haptic displays for scientific visualization. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, F. Baskett, Ed., vol. 24, 177–185.

BURDEA, G. C. 1996. *Force and Touch Feedback for Virtual Reality*. John Wiley and Sons, Inc.

ÇAVUŞOĞLU, M. C., AND TENDICK, F. 2000. Multirate simulation for high fidelity haptic interaction with deformable objects in virtual environments. *Proc. of IEEE International Conference on Robotics and Automation*, 2458–2465.

CHARBONNEAU, E., MILLER, A., WINGRAVE, C., AND LAVIOLA, J. 2009. Understanding visual interfaces for the next generation of dance-based rhythm video games. In *To Appear in Sandbox 2009: ACM SIGGRAPH Video Game Proceedings*.

CHEN, E. 1999. Six degree-of-freedom haptic system for desktop virtual prototyping applications. In *Proceedings of the First International Workshop on Virtual Reality and Prototyping*, 97–106.

CIRAK, F., AND WEST, M. 2005. Decomposition contact response (DCR) for explicit finite element dynamics. *International Journal for Numerical Methods in Engineering 64*, 8.

COLGATE, J. E., AND SCHENKEL, G. G. 1994. Passivity of a class of sampled-data systems: Application to haptic interfaces. *Proc. of American Control Conference*.

COLGATE, J. E., GRAFING, P. E., STANLEY, M. C., AND SCHENKEL, G. 1993. Implementation of stiff virtual walls in force-reflecting interfaces. *Virtual Reality Annual International Symposium*, 202–207.

COLGATE, J. E., STANLEY, M. C., AND BROWN, J. M. 1995. Issues in the haptic display of tool use. *Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 140–145.

CONNER, B. D., SNIBBE, S. S., HERNDON, K. P., ROBBINS, D. C., ZELEZNIK, R. C., AND VAN DAM, A. 1992. Three-dimensional widgets. In *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 183–188.

CONSTANTINESCU, D., SALCUDEAN, S. E., AND CROFT, E. A. 2005. Local model of interaction for realistic manipulation of rigid virtual worlds. *International Journal of Robotics Research 24*, 10.

COTTLE, R., PANG, J., AND STONE, R. 1992. *The Linear Complementarity Problem*. Academic Press.

DAVIS, J., AGRAWALA, M., CHUANG, E., POPOVIC, Z., AND SALESIN, D. 2003. A sketching interface for articulated figure animation. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

DURIEZ, C., ANDRIOT, C., AND KHEDDAR, A. 2004. A multi-threaded approach for deformable/rigid contacts with haptic feedback. *Proc. of Haptics Symposium*.

DURIEZ, C., DUBOIS, F., KHEDDAR, A., AND ANDRIOT, C. 2006. Realistic haptic rendering of interacting deformable objects in virtual environments. *Proc. of IEEE TVCG 12*, 1.

EDMOND, C., HESKAMP, D., SLUIS, D., STREDNEY, D., WIET, G., YAGEL, R., WEGHORST, S., OPPENHEIMER, P., MILLER, J., LEVIN, M., AND ROSENBERG, L. 1997. Ent endoscopic surgical simulator. *Proc. of Medicine Meets VR*, 518–528.

EHMANN, S., AND LIN, M. C. 2001. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum (Proc. of Eurographics'2001) 20*, 3, 500–510.

FEINER, S., MACINTYRE, B., HAUPT, M., AND SOLOMON, E. 1993. Windows on the world: 2d windows for 3d augmented reality. In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, 145–155.

FISHER, B., FELS, S., MACLEAN, K., MUNZNER, T., AND RENSINK, R. 2004. Seeing, hearing and touching: Putting it all together. In *ACM SIGGRAPH course notes*.

GARRE, C., AND OTADUY, M. A. 2009. Haptic rendering of complex deformations through handle-space force linearization. *Proc. of World Haptics Conference*.

GOERTZ, R., AND THOMPSON, R. 1954. Electronically controlled manipulator. *Nucleonics*, 46–47.

GREGORY, A., MASCARENHAS, A., EHMANN, S., LIN, M. C., AND MANOCHA, D. 2000. 6-DoF haptic display of polygonal models. *Proc. of IEEE Visualization Conference*.

GUENDELMAN, E., BRIDSON, R., AND FEDKIW, R. 2003. Nonconvex rigid bodies with stacking. *ACM Trans. on Graphics (Proc. of ACM SIGGRAPH) 22*, 871–878.

HANNAFORD, B., RYU, J.-H., AND KIM, Y. S. 2002. Stable control of haptics. In *Touch in Virtual Environments*, M. L. McLaughlin, J. P. Hespanha, and G. S. Sukhatme, Eds. Prentice Hall PTR, Upper Saddle River, NJ, ch. 3, 47–70.

HARDERS, M., AND SZEKELY, G. 2003. Enhancing human computer interaction in medical segmentation. *Proceedings of the IEEE, Special Issue on Multimodal Human Computer Interfaces 91*, 9, 1430–1442.

HAYWARD, V., AND ARMSTRONG, B. 2000. A new computational model of friction applied to haptic rendering. *Experimental Robotics VI*.

HAYWARD, V., GREGORIO, P., ASTLEY, O., GREENISH, S., AND DOYON, M. 1998. Freedom-7: A high fidelity seven axis haptic device with applications to surgical training. *Experimental Robotics*, 445–456. Lecture Notes in Control and Information Sciences 232.

HILL, J. W., AND SALISBURY, J. K. 1977. Two measures of performance in a peg-in-hole manipulation task with force feedback. *Thirteenth Annual Conference on Manual Control, MIT*.

HOGAN, N. 1985. Impedance control: An approach to manipulation, part i - theory, part ii - implementation, part iii - applications. *Journal of Dynamic Systems, Measurement and Control 107*, 1–24.

HOGAN, N. 1986. Multivariable mechanics of the neuromuscular system. *IEEE Annual Conference of the Engineering in Medicine and Biology Society*, 594–598.

IGARASHI, T., AND HUGHES, J. F. 2002. Clothing manipulation. *Proc. of UIST*.

IGARASHI, T., AND HUGUES, J. F. 2001. A suggestive interface for 3D drawing. *Proc. of UIST*.

IGARASHI, T., MATSUOKA, S., KAWACHIYA, S., AND TANAKA, H. 1997. Interactive Beautification: A technique for rapid geometric design. *Proc. of UIST*, 105–114.

IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3D freeform design. *Proc. of ACM SIGGRAPH*.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-rigid-as-possible shape manipulation. *Proc. of ACM SIGGRAPH*.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. Spatial keyframing for performance-driven animation. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

IJIRI, T., OKABE, M., OWADA, S., AND IGARASHI, T. 2005. Floral diagrams and inflorescences: Interactive flower modeling using botanical structural constraints. *Proceedings of ACM SIGGRAPH*.

INSKO, B., MEEHAN, M., WHITTON, M., AND BROOKS, F. 2001. Passive haptics significantly enhances virtual environments. Tech. Rep. 01-010, Department of Computer Science, UNC Chapel Hill.

INSKO, B. 2001. *Passive Haptics Significantly Enhance Virtual Environments*. PhD thesis, University of North Carolina. Department of Computer Science.

JOHNSON, D. E., AND COHEN, E. 2001. Spatialized normal cone hierarchies. *Proc. of ACM Symposium on Interactive 3D Graphics*, pp. 129–134.

JOHNSON, D. E., AND WILLEMSEN, P. 2003. Six degree of freedom haptic rendering of complex polygonal models. In *Proc. of Haptics Symposium*.

JOHNSON, D. E., AND WILLEMSEN, P. 2004. Accelerated haptic rendering of polygonal models through local descent. *Proc. of Haptics Symposium*.

JOHNSON, D. E., WILLEMSEN, P., AND COHEN, E. 2005. 6-dof haptic rendering using spatialized normal cone search. *IEEE Transactions on Visualization and Computer Graphics 11*, 6, 661–670.

KARNOPP, D. 1985. Computer simulation of stick slip friction in mechanical dynamic systems. *Trans. ASME, Journal of Dynamic Systems, Measurement, and Control*.

KATZOURIN, M., IGNATOFF, D., QUIRK, L., LAVIOLA, J., AND JENKINS, O. C. 2006. Swordplay: Innovating game development through vr. *IEEE Computer Graphics and Applications 26*, 6, 15–19.

KAUFMAN, D. M., EDMUNDS, T., AND PAI, D. K. 2005. Fast frictional dynamics for rigid bodies. *Proc. of ACM SIGGRAPH*.

KHO, Y., AND GARLAND, M. 2005. Sketching mesh deformations. *Proceedings of the ACM Symposium on Interactive 3D Graphics*, 147–154.

KIM, W., AND BEJCZY, A. 1991. Graphical displays for operator aid in telemanipulation. *IEEE International Conference on Systems, Man and Cybernetics*.

KIM, Y. J., LIN, M. C., AND MANOCHA, D. 2002. DEEP: an incremental algorithm for penetration depth computation between convex polytopes. *Proc. of IEEE Conference on Robotics and Automation*, 921–926.

KIM, Y. J., OTADUY, M. A., LIN, M. C., AND MANOCHA, D. 2003. Six-degree-of-freedom haptic rendering using incremental and localized computations. *Presence 12*, 3, 277–295.

KLATZKY, R. L., AND LEDERMAN, S. J. 2003. Touch. In *Experimental Psychology*, 147–176. Volume 4 in I.B. Weiner (Editor-in-Chief). Handbook of Psychology.

LARSEN, E. 2001. A robot soccer simulator: A case study for rigid body contact. *Game Developers Conference*.

LAVIOLA JR., J. J. 2008. Bringing vr and spatial 3d interaction to the masses through video games. *IEEE Comput. Graph. Appl. 28*, 5, 10–15.

LAVIOLA, J. 2000. Msvt: A virtual reality-based multimodal scientific visualization tool. In *Proceedings of the Third IASTED International Conference on Computer Graphics and Imaging*, 1–7.

LAWRENCE, D. A., LEE, C. D., PAO, L. Y., AND NOVOSELOV, R. 2000. Shock and vortex visualization using a combined visual/haptic interface. *Proc. IEEE Visualization*, 131–137.

LEE, J. 2008. Hacking the nintendo wii remote. *Pervasive Computing, IEEE 7*, 3 (July-Sept.), 39–45.

LIN, M. C., AND OTADUY, M. A. 2008. *Haptic Rendering: Foundations, Algorithms & Applications*. AK Peters.

LUK, J., PASQUERO, J., LITTLE, S., MACLEAN, K. E., HAYWARD, V., AND LEVESQUE, V. 2006. Haptics as a solution for mobile interaction challenges: Initial design using a handheld tactile display prototype. *Proceedings of ACM Conference on Human Factors in Computing Systems, CHI*.

MAPES, D., AND MOSHELL, M. 1995. A two-handed interface for object manipulation in virtual environments. *Presence: Teleoper. Virtual Environ. 4*, 4, 403–416.

MARK, W., RANDOLPH, S., FINCH, M., VAN VERTH, J., AND TAYLOR II, R. M. 1996. Adding force feedback to graphics systems: Issues and solutions. In *SIGGRAPH 96 Conference Proceedings*, 447–452.

MARK, W., RANDOLPH, S., FINCH, M., VAN VERTH, J., AND TAYLOR II, R. M. 1996. Adding force feedback to graphics systems: Issues and solutions. In *SIGGRAPH 96 Conference Proceedings*, H. Rushmeier, Ed., Annual Conference Series, 447–452.

MASSIE, T. M., AND SALISBURY, J. K. 1994. The phantom haptic interface: A device for probing virtual objects. *Proc. of ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems 1*, 295–301.

MCLAUGHLIN, M., HESPANHA, J. P., AND SUKHATME, G. S. 2002. *Touch in Virtual Environments*. Prentice Hall.

MCNEELY, W., PUTERBAUGH, K., AND TROY, J. 1999. Six degree-of-freedom haptic rendering using voxel sampling. *Proc. of ACM SIGGRAPH*, 401–408.

MCNEELY, W., PUTERBAUGH, K., AND TROY, J. 2006. Voxel-based 6-dof haptic rendering improvements. *Haptics-e 3*, 7.

MILENKOVIC, V. J., AND SCHMIDL, H. 2001. Optimization-based animation. *SIGGRAPH 01 Conference Proceedings*, 37–46.

MILLER, B. E., COLGATE, J. E., AND FREEMAN, R. A. 1999. Guaranteed stability of haptic systems with nonlinear virtual environments. *IEEE Transactions on Robotics and Automation 16*, 6, 712–719.

MINE, M. R., BROOKS, JR., F. P., AND SEQUIN, C. H. 1997. Moving objects in space: exploiting proprioception in virtual-environment interaction. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 19–26.

MINE, M. 1995. Virtual environment interaction techniques. Tech. rep., UNC Chapel Hill CS Dept.

MINSKY, M., OUH-YOUNG, M., STEELE, O., BROOKS, JR., F. P., AND BEHENSKY, M. 1990. Feeling and seeing: Issues in force display. In *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, R. Riesenfeld and C. Sequin, Eds., vol. 24, 235–243.

MIRTICH, B. V. 1996. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California, Berkeley.

MIRTICH, B. 2000. Timewarp rigid body simulation. *SIGGRAPH 00 Conference Proceedings*, 193–200.

MORI, Y., AND IGARASHI, T. 2007. Plushie: An interactive design system for plush toys. *Proc. of ACM SIGGRAPH*.

MÜLLER, M., AND GROSS, M. 2004. Interactive virtual materials. *Proc. of Graphics Interface*.

MÜLLER, M., DORSEY, J., MCMILLAN, L., JAGNOW, R., AND CUTLER, B. 2002. Stable real-time deformations. *Proc. of ACM SIGGRAPH Symposium on Computer Animation*.

NEALEN, A., SORKINE, O., ALEXA, M., AND COHEN-OR, D. 2005. A sketch-based interface for detail-preserving mesh editing. *Proc. of ACM SIGGRAPH*.

NEALEN, A., IGARASHI, T., SORKINE, O., AND ALEXA, M. 2007. FiberMesh: Designing freeform surfaces with 3D curves. *Proc. of ACM SIGGRAPH*.

NELSON, D. D., JOHNSON, D. E., AND COHEN, E. 1999. Haptic rendering of surface-to-surface sculpted model interaction. *Proc. of ASME Dynamic Systems and Control Division*.

OKABE, M., OWADA, S., AND IGARASHI, T. 2005. Interactive design of botanical trees using freehand sketches and example-based editing. *Proceedings of Eurographics*.

ORTEGA, M., REDON, S., AND COQUILLART, S. 2006. A six degree-of-freedom god-object method for haptic display of rigid bodies. *Proc. of IEEE Virtual Reality Conference*.

OTADUY, M. A., AND GROSS, M. 2007. Transparent rendering of tool contact with compliant environments. *Proc. of World Haptics Conference*.

OTADUY, M. A., AND LIN, M. C. 2005. Stable and responsive six-degree-of-freedom haptic manipulation using implicit integration. *Proc. of World Haptics Conference*, 247–256.

OTADUY, M. A., AND LIN, M. C. 2006. A modular haptic rendering algorithm for stable and transparent 6-DOF manipulation. *IEEE Transactions on Robotics 22*, 4, 751–762.

OTADUY, M. A., TAMSTORF, R., STEINEMANN, D., AND GROSS, M. 2009. Implicit contact handling for deformable objects. *Proc. of Eurographics*.

OTADUY, M. A. 2004. *6-DoF Haptic Rendering Using Contact Levels of Detail and Haptic Textures*. PhD thesis, Department of Computer Science, University of North Carolina at Chapel Hill.

OUH-YOUNG, M. 1990. *Force Display in Molecular Docking*. PhD thesis, University of North Carolina, Computer Science Department.

OWADA, S., NIELSEN, F., NAKAZAWA, K., AND IGARASHI, T. 2003. A sketching interface for modeling the internal structures of 3D shapes. *Smart Graphics*.

PAUSCH, R., BURNETTE, T., BROCKWAY, D., AND WEIBLEN, M. E. 1995. Navigation and locomotion in virtual worlds via flight into hand-held miniatures. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, 399–400.

PICINBONO, G., LOMBARDO, J.-C., DELINGETTE, H., AND AYACHE, N. 2000. Anisotropic elasticity and forces extrapolation to improve realism of surgery simulation. *Proc. of IEEE International Conference on Robotics and Automation*.

PIERCE, J. S., FORSBERG, A. S., CONWAY, M. J., HONG, S., ZELEZNIK, R. C., AND MINE, M. R. 1997. Image plane interaction techniques in 3d immersive environments. In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, ACM, New York, NY, USA, 39–ff.

POUPYREV, I., BILLINGHURST, M., WEGHORST, S., AND ICHIKAWA, T. 1996. The go-go interaction technique: non-linear mapping for direct manipulation in vr. In *UIST '96: Proceedings of the 9th annual ACM symposium on User interface software and technology*, ACM, New York, NY, USA, 79–80.

REKIMOTO, J. 2002. SmartSkin: An infrastructure for freehand manipulations on interactive surfaces. *Proceedings of CHI*, 113–120.

RUSPINI, D., KOLAROV, K., AND KHATIB, O. 1997. The haptic display of complex graphical environments. *Proc. of ACM SIGGRAPH*, 345–352.

SALISBURY, K., BROCK, D., MASSIE, T., SWARUP, N., AND ZILLES, C. 1995. Haptic rendering: Programming touch interaction with virtual objects. In *1995 Symposium on Interactive 3D Graphics*, P. Hanrahan and J. Winget, Eds., ACM SIGGRAPH, 123–130. ISBN 0-89791-736-7.

SCHMIDT, R., WYVILL, B., SOUSA, M. C., AND JORGE, J. A. 2005. ShapeShop: Sketch-based solid modeling with BlobTrees. *Eurographics Workshop on Sketch-based Interfaces and Modeling*.

SHABANA, A. A. 1989. *Dynamics of Multibody Systems*. John Wiley and Sons.

SHIMOGA, K. 1992. Finger force and touch feedback issues in dextrous manipulation. *NASA-CIRSSE International Conference on Inetelligent Robotic Systems for Space Exploration*.

SILVA, M. G., AND BOWMAN, D. A. 2009. Body-based interaction for desktop games. In *CHI EA '09: Proceedings of the 27th international conference extended abstracts on Human factors in computing systems*, ACM, New York, NY, USA, 4249–4254.

SIROUSPOUR, M. R., DIMAIO, S. P., SALCUDEAN, S. E., ABOLMAESUMI, P., AND JONES, C. 2000. Haptic interface control - design issues and experiments with a planar device. *Proc. of IEEE International Conference on Robotics and Automation*.

SLATER, M., AND USOH, M. 1993. An experimental exploration of presence in virtual environments. Tech. Rep. 689, Department of Computer Science, University College London.

STEWART, D. E., AND TRINKLE, J. C. 2000. An implicit time-stepping scheme for rigid body dynamics with coulomb friction. *IEEE International Conference on Robotics and Automation*, 162–169.

STOAKLEY, R., CONWAY, M. J., AND PAUSCH, R. 1995. Virtual reality on a wim: interactive worlds in miniature. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 265–272.

SUTHERLAND, I. 1965. The ultimate display. *Proc. of IFIP*, 506–508.

THORNE, M., BURKE, D., AND PANNE, M. 2004. Motion Doodles: An interface for sketching character motion. *Proc. of ACM SIGGRAPH*.

TURQUIN, E., CANI, M.-P., AND HUGHES, J. F. 2004. Sketching garments for virtual characters. *Eurographics Workshop on Sketch-based Interfaces and Modeling*.

UNGER, B. J., NICOLAIDIS, A., BERKELMAN, P. J., THOMPSON, A., LEDERMAN, S. J., KLATZKY, R. L., AND HOLLIS, R. L. 2002. Virtual peg-in-hole performance using a 6-dof magnetic levitation haptic device: Comparison with real forces and with visual guidance alone. *Proc. of Haptics Symposium*, 263–270.

WAN, M., AND MCNEELY, W. A. 2003. Quasi-static approximation for 6 degrees-of-freedom haptic rendering. *Proc. of IEEE Visualization*, 257–262.

WLOKA, M. M., AND GREENFIELD, E. 1995. The virtual tricorder: a uniform interface for virtual reality. In *UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology*, ACM, New York, NY, USA, 39–40.

WU, D. 2000. Penalty methods for contact resolution. *Game Developers Conference*.

ZELEZNIK, R. C., HERNDON, K. P., AND HUGHES, J. F. 1996. SKETCH: An interface for sketching 3D scenes. *Proc. of ACM SIGGRAPH*.

ZIENKIEWICZ, O. C., AND TAYLOR, R. L. 1989. *The Finite Element Method*, 4th ed. McGraw-Hill.

ZILLES, C., AND SALISBURY, K. 1995. A constraint-based god object method for haptics display. In *Proc. of IEEE/RSJ Int. Conf. on Intelligent Robotics and Systems*.