Interactive Shadows

Kenneth P. Herndon, Robert C. Zeleznik,
Daniel C. Robbins, D. Brookshire Conner,
Scott S. Snibbe and Andries van Dam

Brown University
PO Box 1910
Providence, RI 02912
(401) 863-7693
{kph,bcz,dcr,dbc,sss,avd}@cs.brown.edu

ABSTRACT

It is often difficult in computer graphics applications to understand spatial relationships between objects in a 3D scene or effect changes to those objects without specialized visualization and manipulation techniques. We present a set of three-dimensional tools (widgets) called "shadows" that not only provide valuable perceptual cues about the spatial relationships between objects, but also provide a direct manipulation interface to constrained transformation techniques. These shadow widgets provide two advances over previous techniques. First, they provide high correlation between their own geometric feedback and their effects on the objects they control. Second, unlike some other 3D widgets, they do not obscure the objects they control.

Keywords

Direct Manipulation, 3D Widgets, Interactive Systems

1 Introduction

A wide variety of techniques for visualizing and manipulating objects have been implemented in interactive 3D graphics applications for modeling, animation, simulation and visualization. In this paper, we present a set of widgets[1] called *shadows* that we have used both to view and to interact with objects in our 3D application environment. These widgets are similar to shadows implemented in previous systems in displaying information about a scene's geometric composition. However, the shadow widgets described here are interactive extensions of this idea — they allow users to translate, rotate and scale objects in a constrained manner. We discuss the details of our implementation of these *shadow* widgets and some of the problems associated with them.

2 Previous Work

2.1 Interaction Techniques

Almost all techniques for visualizing and manipulating objects in 3D computer graphics applications have been developed for hardware configurations that include 2D input devices such as mice or tablets and conventional CRT displays (i.e., non-stereo and non-immersive displays). It has thus been necessary to give the user tools to map both 2D user input to 3D object manipulation and 3D spatial relationships to 2D displays. Two-dimensional widgets, such as sliders or virtual dials, can be used to apply changes to an object (translation, color, etc.), but these widgets generally are external to the 3D scene. Thus, there is often a large cognitive "distance" between a user's intentions and an application's tools; if this distance is too great, the kinesthetic correspondence between user actions and results will be poor and the user's sense of engagement with the application is lessened [11].

Direct-manipulation techniques, or widgets, that significantly reduce the cognitive load on a user in certain interaction tasks have been implemented in many systems [5] [10] [15] [16] [22]. These widgets exist within a 3D scene along with the very objects they affect, thus increasing a user's feeling of performing actions directly upon objects and decreasing the distance between goals and actions. However, none of these solutions are perfect for all interaction tasks [4] [14].

Adding geometry to a widget sometimes helps communicate the widget's degrees of freedom and intended use, thus allowing a user to more accurately choose the appropriate tool for a task and predict that tool's effects. However, geometric 3D widgets can still distract a user's attention away from or even obscure the object of interest [21]. Alleviating these problems by rendering widgets in wireframe or disclosing their components selectively [22], can make it more difficult to understand the function of a widget or its relation to a 3D object.

2.2 Visualization Techniques

The 2D image produced by hardware Z-buffer renderers (the most common style of representation) is often hard to visualize, and a great deal of viewpoint or object manipulation is required to form a mental model of a scene. A scene's complexity increases when a user needs to visualize different attributes of a model simultaneously, such as spatial or logical relationships between its parts, or simply visual properties like surface color and texture. Although these qualitatively different ways to look at a model are often treated as different modules of the same application (or even different applications), they are all related through the underlying model itself. Thus, many commercial and research applications support multiple window displays to visualize simultaneously different parts, views, or representations of a model. While this solution does present a great deal of information at once, it is the user who must combine the separate images into a single, coherent understanding of the model. This is an often difficult task [6]; consider combining three adjacent orthographic views and one perspective view, a standard multi-window configuration, to form a mental model of a complex object. Moreover, when the user wishes to interact with the model,

[1] A *widget* is an interaction technique that encapsulates geometry and behavior and is used to visualize and/or control application objects.

she must choose the one view, or worse the set of views, most appropriate for the task.

One technique sometimes used to visualize geometric relationships between parts of a 3D model is an orthogonal projection of objects onto a floor or wall plane. Such a shadow, so long as it does not lie along an axis similar to the viewing direction, provides important information about the scene. Unfortunately, most implementations of this idea are non-interactive (i.e., users cannot interact directly with the shadows themselves); the shadows are exclusively visualization aids.

The GROPE-II system, developed at the University of North Carolina at Chapel Hill in the mid-1970s [12], provided a number of depth cues, including shadows, to help users position and orient a robot's arm and hand in a real environment by interacting with a simulated representation on a vector display. User studies conducted by the author revealed that shadows were the most informative and popular cue. The *bolio* system from MIT [23], and Jack, by Phillips and Badler [15], also projected modeling objects onto the walls of the scene to aid visualization. Projections of data points in 3D volumes have also been used in scientific visualization applications to help users understand their data [9].

Wanger et al. recently performed user studies to determine which of a number of depth cues, including shadows, most effectively displayed inter-object spatial relationships to a user viewing a scene via a 2D display. Shadows ranked high in these results [25] [26].

In none of these systems or studies are users allowed to manipulate the shadows themselves — the projections are useful visualization aids, but are non-interactive. However, in a recent system, specular highlights and shadows derived from light sources are used to position lights in a 3D scene [18]. In this system, light sources are inferred from information provided by user-specified highlights and shadows.

3 Using Shadows as Widgets

Projections of objects (or shadows) have been proven to be valuable visualization tools. We have not only used them extensively for this purpose, but have also allowed users to interact with shadows to apply transformations to objects.

Figure 1: By dragging the shadow of the cube object, the cube itself is translated by the same amount in a plane parallel to the shadow plane.

3.1 What is a Shadow Widget?

Our interactive shadow widget is a projection of a 3D object onto a plane (usually aligned with the 3D coordinate system of the world and located near the 3D object). It is related to the 3D object itself via a network of two-way dependencies that "connect" the 3D object and its projection so that they translate, rotate and scale with each other. A user can transform the 3D object freely in three-space using any of a variety of direct-manipulation techniques and watch its shadow move accordingly. Conversely, dragging an object's shadow moves the 3D object (Figure 1). This shadow is constrained to lie in a single plane, and transformation of the shadow widget affects the 3D object in that same plane. Translation and scale occur in the projection plane and rotation occurs about the normal to the projection plane. Because more than one shadow widget can be activated at a time for the same 3D object, shadows of a single object can be placed on each of three separate orthogonal planes (Figures 2 and 3). Also, each object may have its own set of corresponding shadow widgets, although to avoid clutter in a multiple-object scene, we typically use shadows only for objects being manipulated.

Figure 2: A model of an airplane with detached landing gear is displayed in a stage. Opaque shadows of the model parts are projected onto the floor plane and both walls.

Figure 3: This schematic labels the elements of Figure 2.

3.2 Different Types of Shadows

The geometry of a shadow widget is usually directly related to the geometry of the 3D object it comes from. Since shadow widgets are first-class objects in our system, their attributes can be altered to produce different effects. For instance, we have used different rendering styles, including opaque black silhouettes, wireframe, and fully rendered copies of the 3D object, to produce shadow widgets.

We can also alter the geometry of the shadow in situations where it is less important than other properties such as position or color.

The opaque version of the shadow widget most closely resembles our real-world experience of shadows produced by distant light sources. The silhouette appears as an accurate geometric profile of the object devoid of any surface features. In a scene containing a number of differently shaped objects and their shadows, users can generally associate an object with its shadow and use this knowledge to understand spatial relationships between objects in the scene. However, difficulties can arise with certain object-shadow relationships when, for example, the projections of two very differently shaped objects are similar from one point of view (say, a sphere and an hourglass) [25]. In these cases, projecting shadows simultaneously onto a number of orthogonal planes can help disambiguate conflicts.

Wireframe shadows are similar to the orthogonal views commonly used in industrial design or architectural applications. These projections allow users to "see through" objects and thus perceive spatial relationships among them even when the objects intersect one other (Figure 4). Wireframe views can be used to align surface and interior features of objects precisely with one another.

Figure 4: The shadow objects may be rendered in wire-frame to display relationships between interpenetrating objects.

Fully rendered shadows resemble real-world mirrors in that they display the surface properties of the actual 3D object (color, shading, texture, etc.). This type of shadow is useful when more visual information about the object of interest is required than either opaque or wireframe shadows can provide.

Shadows may also be created with arbitrary geometry. This style of shadow widget still reflects attributes of the primary 3D object, but need not be a geometrically exact projection. In some cases, it may be useful to substitute an alternate geometric shape for the projection of the object's actual geometry. If, for instance, the scene being visualized contains a large number of relatively complex objects, geometrically accurate shadows may be either unnecessary or prohibitively expensive to render, and a simple bounding circle or square may suffice to depict the spatial relationships. Alternatively, textual names of objects may be projected into the shadow plane, providing labels for the model that do not themselves obscure the geometry of the model.

These different styles of shadow widgets can be used simultaneously. For example, a fully rendered shadow of an object can be put next to wireframe shadows of surrounding objects to stress the importance of one object in the scene over others. In addition, the shadows of selected objects can be rendered differently from those of unselected objects.

3.3 Use of Shadows

A single shadow widget by itself is useful both as a spatial cue and as a tool to constrain transformations to a plane. In practice, however, we use multiple shadows set up in a stage format. In this configuration, the 3D object is enclosed in a larger cube. Shadow widgets are projected onto the interior walls of this cube, but are rendered only for walls that lie behind the object of interest with respect to the viewpoint. This stage metaphor has been implemented in a number of previous systems to give users some sense of a workspace in an otherwise boundless volume [8] [15].

Displaying a number of shadow widgets simultaneously in a stage-like configuration like this has two important implications. First, users are easily able to transform objects with planar constraints using controls that are readily available and visible at all times. Other solutions to this specific interaction task include widgets known as object handles [5] [10] [15] [17] [22]. These handles are useful tools in many situations, but their geometry can sometimes obscure the object of interest. This can make it difficult for users to see the changes they have made to an object. For example, aligning two objects with one another can be very difficult if the tool being used conceals the interesting features of the objects themselves. Shadow widgets may be used in such situations to remove the obscuring geometry of the tool from the object being manipulated without completely breaking the visual connection between tool and object.

Second, using shadows on a stage provides visualization and interaction functionality comparable to multiple-window solutions, including interaction with 3D objects from three orthogonal and one perspective view, with the advantage that the views are integrated in a single 3D view. In this view, the position and orientation of the shadows are always consistent with the actual 3D objects, since all are views of the same underlying model. We have found in this setup that, even though the projected views are generally seen at an angle, we are still able to use them effectively as orthogonal views.

Other perceptual advantages to using shadow widgets for both manipulation and visualization derive from the fact that the shadow widgets rely on a concrete visual metaphor (i.e., their relation to real-world phenomena) to convey their function to a user: real-world shadows move along the surfaces of objects; shadow widgets, similarly, move in a plane. The shadow widgets do not, however, adequately reveal their interactive functionality — users must be told that they can move the shadow directly and that doing so moves the related 3D object. User studies are needed to determine just how much better or worse these widgets are that others for interaction with objects.

This method of object manipulation and visualization is obviously not desirable for all interactive tasks, but we have found it useful for some. Shadows provide an alternate means for viewing and interacting with a scene from different points of view that is perceptually more unified than multiple-window solutions. In any graphics application, we believe that the user should have available a variety of tools, and should be able to switch between manipulation and visualization tools easily, choosing the one best fitting the specific task at hand. Shadows are one such tool.

3.4 Extending the Use of Shadows

The interactive shadow widget, when displayed as a fully rendered object, has all the surface properties of the 3D object from which it is derived. We call this version of the shadow widget a "mirror" widget. When the widget is placed behind a 3D object (with respect to the user's point of view), it is flipped around the projection axis, so that the user can see the back sides of the object (Figure 5).

Like shadows, these "mirrors" do not act at all like their real-world analogues — a shadow widget is not computed from a light source, and these "mirrors" are not view-dependent.

Figure 5: Fully-rendered shadows are used to display surfaces of objects that are not directly visible from the user's point of view. Note that in the mirror view, one can see that the landing gear is aligned with the landing gear bay.

We have also used this specific version of the shadow widget to perform object-to-object snapping [1] [2], a popular technique for aligning the surfaces of two objects with each other. In our implementation of this technique, we can snap together only points on the visible surfaces of objects. Frequently, one or more of the surfaces to be snapped is not visible from the user's point of view, and either the viewpoint or the model must be altered to reveal it. If a user is performing a number of similar snapping tasks in sequence, she must interleave snapping operations with viewpoint modification operations. This constant interleaving of tasks can be distracting and time-consuming.

We can take advantage of the additional visual information provided by the fully rendered shadow widgets when performing object-to-object snapping between surfaces that are not simultaneously visible. Consider a scene in which one surface to be snapped is directly visible to the user, and the other is not. An appropriately placed "mirror" widget will reveal (in "reflection") the invisible surface, and thus increase the number of visible surfaces in the scene. Clicking on the "mirror" image of the previously invisible surface records the corresponding point on the actual surface of the 3D object. In a scene with more than one fully-rendered shadow, snapping operations can occur between any two "mirror" images, or between any "mirror" and object. This method works best for objects that are completely convex. Objects that have self-obscuring regions may not benefit fully from the use of full-rendered shadow widgets.

4 Implementation Details and Problems

We have used our animation, modeling and simulation system, called UGA [27], to implement the shadow widgets described here. This system uses the same scripting language to describe the geometry and behavior of widgets and application objects, and treats both equally as first-class objects. Shadow widgets are created by first copying the geometry of a 3D object, then scaling the copy along the axis perpendicular to the projection plane to a very thin size, and translating it by some amount to offset it from the primary object (Figure 6). Finally, a two-way constraint is defined between the transformation matrices of the object and its shadow to assure that changes to either object affects the other.

Figure 6: The airplane model (here in 2D) is copied, scaled down along the projection axis, and colored black to create an opaque shadow. Notice how the shadow object retains some thickness, and is offset from the floor plane. The scale of the shadow and its distance from the floor plane are exaggerated in this figure.

We do not scale the copies to zero along the projection axis because of our pick-correlation method. Objects are picked by casting a ray into the scene from the viewpoint through the mouse cursor position. This ray is intersected with each object in the scene to determine which object was clicked on. The ray intersection test for each object is performed in object space, which requires inversion of the object's transformation matrix, and is impossible for a zero-scaled matrix (see [7], Chapter 15, for more details on ray tracing).

The trick of thinly scaling objects relieves us of the computational costs of constructing shadow volumes and intersecting them with the shadow plane, at the expense of some generality (for instance, our shadow widgets can "fall" only on planar surfaces). However, this method has drawbacks. For instance, in a scene with many objects and shadow widgets, it is important to ensure that the shadow widgets for different objects do not all lie in exactly the same plane, especially if they are fully rendered. Z-buffer renderers produce visible errors when rendering coplanar polygons. Placing shadows of different objects in slightly different planes also brings up the issue of the order they should be in. It would make sense to place shadows for objects closest to the projection plane "above" shadows for objects further away. This method is necessary mainly for fully rendered shadows; the order of opaque black shadows may seem irrelevant because their surface details are practically invisible. However, for pick correlation, the order may indeed be important.

The opaque version of the shadow widget is simply produced by the above method and colored black — all of the polygons of the original object are preserved, but their color is overwritten. The wireframe shadows are copies of the original objects as well, and are merely rendered differently from the original shaded model. Fully rendered shadows are scaled negatively to invert the polygonal copy of the original object along the scaling axis. In this negatively scaled copy, the surface normals for polygons are scaled by the same transformation matrix as the polygons themselves. This can have the effect of "flattening out" the shading of the copy, thus decreasing or eliminating any surface definition. An alternate method might not scale the surface normals to avoid this potential problem. In either case, the shading for a fully rendered shadow widget is computed by the Z-buffer renderer after the shadow has been created and placed

in the scene. Thus, a fully rendered shadow widget may not look exactly like a reflection of the non-visible side of an object because it is technically not a mirror image of that object.

The fully rendered shadow widgets present some complications when used for object-to-object snapping. We determine snapping points by intersecting rays with objects in the scene. When a user clicks on a fully rendered shadow widget, the ray-intersection algorithm naturally returns information about the surface of the widget object. Obviously, it is not the surface information of the shadow that we want, but rather that of the 3D object to which the shadow is related. To get the surface information we want, we place between the fully rendered shadows and the 3D objects, a thinly scaled, semi-transparent cube that intercepts rays cast from the mouse location and "bounces" them back along the shadow widget's axis of projection. This redirected ray then intersects the 3D object and returns the surface information at this intersection point for use by the snapping technique. However, the semitransparent cube that redirects rays is slightly offset from the projection plane of the shadow widgets so that it does not interfere with rendering. Consequently, from certain angles, pick correlation for points on the shadow widgets may be somewhat inaccurate (Figure 7).

Figure 7: This figure demonstrates one problem with our implementation of pick-correlation with fully-rendered shadows. A "mirror" object is placed between the shadows of objects A and B which intercepts incoming rays from the viewpoint. In this diagram, the user is attempting to select object B by clicking on its shadow. The floor plane intercepts this ray and "reflects" it along its normal vector. In this case, object A will be selected.

Shadow widgets, as we have implemented them, present efficiency problems because they are exact duplicates of polygonal objects in the scene. For each new shadow widget introduced, the total number of polygons in the scene increases by the number of polygons in the object being shadowed. This can quickly reduce a system to less than interactive speed if even somewhat complex models are being displayed. Back-face culling both the objects and shadows can help, but only to a certain extent. Also, as mentioned earlier, alternative geometries can be used for the shadow widgets, since the geometry of a shadow widget need not exactly match that of a 3D object in order for users to grasp the relationship between them.

Another way to speed up the generation of shadows might be to use a Z-buffer to render bitmap images of 3D objects from the point of view of the shadow plane. These bitmaps could then be texture-mapped onto a floor or wall plane and displayed in the appropriate location next to the 3D objects. Unfortunately, few practical hardware architectures are able to support this technique.

Like most interaction techniques, the shadow widgets presented here are also somewhat view-dependent. Indeed, it can happen that from certain viewing angles, a shadow will be displayed edge-on, making interactive translation, rotation or scaling impossible.

5 Future Work

Our shadow widgets address a number of interaction and visualization needs of three-dimensional graphics applications, but they have many possible extensions. First, we would like to explore how best to make these shadow widgets disclose their interactive functionality to users. Also, we would like to exploit the idea that shadow widgets can be regarded as alternative views of a scene that are themselves embedded in the scene. Such embedded or alternate views have been used in other systems, like MCC's Mirage [24], for visualizing slicing planes of scientific data, for drawing floor plans of architectural projects, for displaying hierarchy trees and logical or schematic diagrams of complex objects like electronic assemblies, and for displaying other abstractions of a model. Placing these alternate views in proximity to the actual model, as shadow widgets are placed, may provide visual cues about relationships within the object or between objects that are otherwise difficult to visualize. We hope to explore some of these various uses of alternate embedded views in our own applications.

We are also interested in studying the applicability of these techniques in environments that use 6D input devices like a Polhemus 3Space Isotrack [19] or VPL DataGlove [28]. Immersive environments with stereoscopic displays are also an interesting avenue of research. In such an environment, the shadow widgets are likely to be useful more as constrained manipulation tools than visualization aids because the stereoscopic display itself may be adequate to convey spatial relationships between objects.

User studies must be performed on all of these techniques, already implemented or proposed, to determine their actual usefulness in real-world applications. It will also be useful to determine to what extent geometry can be omitted before a 3D object and its shadow no longer appear to be related to one another. Some studies have already been done on this topic [26], but more are needed.

6 Conclusion

We have presented a technique called shadows for visualizing and manipulating objects in three-dimensional graphics applications. These widgets can combine a number of distinct visual representations of an object in one coherent view, and thus offer a unified environment for visualizing the different geometric and non-geometric properties of a model. Shadows are also useful for object manipulation tasks because they provide controls for constrained transformations without obscuring the object of interest. These widgets solve some of the problems encountered with more traditional methods for visualization and direct manipulation of objects by integrating multiple windows and eliminating the need for frequent camera manipulation.

Acknowledgments

This work was supported in part by NSF/DARPA, IBM, Sun Microsystems, NCR, Hewlett Packard and Digital Equipment Corporation. Software contributions by Bitstream Inc. are gratefully acknowledged. We would also like to thank Nate Huang for technical support and the reviewers for their valuable suggestions.

References

[1] Eric A. Bier. Snap-dragging in three dimensions. In Rich Riesenfeld and Carlo Séquin, editors, *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, pages 193–204. ACM SIGGRAPH, March 1990.

[2] Eric A. Bier and Maureen C. Stone. Snap-dragging. In David C. Evans and Russell J. Athay, editors, *SIGGRAPH '86*

Conference Proceedings, pages 233–240. ACM SIGGRAPH, Addison-Wesley, July 1986.

[3] Edwin E. Catmull, editor. *SIGGRAPH '92 Conference Proceedings*. ACM SIGGRAPH, Addison-Wesley, July 1992.

[4] Michael Chen, S. Joy Mountford, and Abigail Sellen. A study in interactive 3-D rotation using 2-D control devices. In John Dill, editor, *SIGGRAPH '88 Conference Proceedings*, pages 121–129. ACM SIGGRAPH, Addison-Wesley, August 1988.

[5] D. Brookshire Conner, Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, Robert C. Zeleznik, and Andries van Dam. Three-dimensional widgets. In Levoy and Catmull [13], pages 183–188.

[6] C. N. Cooper and R. N. Shepard. Turning something over in the mind. *Scientific American*, 251(6):106–114, 1984.

[7] James D. Foley, Andries van Dam, Steven Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 2nd edition, 1990.

[8] Tinsley A. Galyean and John F. Hughes. Sculpting: An interactive volumetric modeling technique. In Sederberg [20], pages 267–274.

[9] Stanley L. Grotch. Three-dimensional and stereoscopic graphics for scientific data display and analysis. *IEEE Computer Graphics and Applications*, 3(8):31–43, November 1983.

[10] Stephanie Houde. Iterative design of an interface for easy 3D direct manipulation. In *Proceedings of ACM CHI'92 Conference on Human Factors in Computing Systems*, pages 135–142, 1992.

[11] Edwin L. Hutchins, James D. Hollan, and Donald A. Norman. Direct manipulation interfaces. In *Human-Computer Interaction*, volume 1, pages 311–338. Laurence Erlbaum Associates, Inc., 1985.

[12] Paul Jerome Kilpatrick. *The Use of a Kinesthetic Supplement in an Interactive Graphics System*. PhD thesis, University of North Carolina at Chapel Hill, 1976.

[13] Marc Levoy and Edwin E. Catmull, editors. *Proceedings of the 1992 Symposium on Interactive Three-Dimensional Graphics*. ACM SIGGRAPH, March 1992.

[14] Gregory M. Nielson and Dan R. Olson Jr. Direct manipulation techniques for 3D objects using 2D locator devices. In Frank Crow and Stephen M. Pizer, editors, *Proceedings of the 1986 Workshop on Interactive 3D Graphics*, pages 175–182. ACM SIGGRAPH and ACM SIGCHI, October 1986.

[15] Cary B. Phillips and Norman I. Badler. Jack: a toolkit for manipulating articulated figures. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software*, pages 221–229, 1988.

[16] Cary B. Phillips, Norman I. Badler, and John Granieri. Automatic viewing control for 3D direct manipulation. In Levoy and Catmull [13], pages 71–74.

[17] Pixar, Inc. RenderMan Showplace. Macintosh application.

[18] Pierre Poulin and Alain Fournier. Lights from highlights and shadows. In Levoy and Catmull [13], pages 31–38.

[19] F. Rabb, E. Blood, R. Steiner, and H. Jones. Magnetic position and orientation tracking system. *IEEE Transaction on Aerospace and Electronic Systems*, 15(5):709–718, September 1979.

[20] Thomas W. Sederberg, editor. *SIGGRAPH '91 Conference Proceedings*. ACM SIGGRAPH, Addison-Wesley, July 1991.

[21] Scott S. Snibbe, Kenneth P. Herndon, Daniel C. Robbins, D. Brookshire Conner, and Andries van Dam. Using deformations to explore 3D widget design. In Catmull [3], pages 351–352. Video paper.

[22] Paul S. Strauss and Rikk Carey. An object-oriented 3D graphics toolkit. In Catmull [3], pages 341–349.

[23] David J. Sturman, David Zeltzer, and Steve Pieper. Hands-on interaction with virtual environments. In *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 19–24, 1989.

[24] Mark A. Tarlton and P. Nong Tarlton. A framework for dynamic visual applications. In Levoy and Catmull [13], pages 161–164.

[25] Leonard R. Wanger. The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In Levoy and Catmull [13], pages 39–42.

[26] Leonard R. Wanger, James A. Ferwerda, and Donald P. Greenberg. Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications*, 12(3):44–58, May 1992.

[27] Robert C. Zeleznik, D. Brookshire Conner, Matthias W. Wloka, Daniel G. Aliaga, Nate T. Huang, Phillip M. Hubbard, Brian Knep, Henry Kaufman, John F. Hughes, and Andries van Dam. An object-oriented framework for the integration of interactive animation techniques. In Sederberg [20], pages 105–112.

[28] Thomas G. Zimmerman, Jaron Lanier, Chuck Blanchard, Steve Bryson, and Young Harvill. A hand gesture interface device. In *Proceedings of ACM CHI+GI'87 Conference on Human Factors in Computing Systems and Graphics Interface*, pages 189–192, 1987.