Virtual Reality Performance for Virtual Geometry

Robert A. Cross and Andrew J. Hanson
Department of Computer Science
Indiana University
Bloomington, IN 47405

Abstract

We describe the theoretical and practical visualization issues solved in the implementation of an interactive real-time four-dimensional geometry interface for the CAVE, an immersive virtual reality environment. While our specific task is to produce a "virtual geometry" experience by approximating physically correct rendering of manifolds embedded in four dimensions, the general principles exploited by our approach reflect requirements common to many immersive virtual reality applications, especially those involving volume rendering. Among the issues we address are the classification of rendering tasks, the specialized hardware support required to attain interactivity, specific techniques required to render 4D objects, and interactive methods appropriate for our 4D virtual world application.

1 Introduction

In this paper we describe how we have combined general requirements for a broad class of virtual reality applications with the capabilities of special-purpose graphics hardware to support an immersive virtual reality application for viewing and manipulating four-dimensional objects. We present general issues concerning the application of virtual reality methods to scientific visualization, discuss how the resulting requirements are reflected in fundamental rendering tasks, and point out where hardware features have crucial roles to play. The proving ground for our general observations is the design and implementation of an application for visualizing a 4D mathematical world through interaction with 3D volume images. We introduce a task independent rendering paradigm through which, with proper hardware support, we can produce complex realistic images at interactive speeds.

4D Visualization. There have been a variety of systems devoted to the general problem of 4D visualization, ranging from the classic work of Banchoff [2, 1] to geometry viewers such as Geomview [17], our local "MeshView" 4D surface viewer, and specialized high-performance interactive systems such as that of Banks [3]. Our virtual-reality-oriented work builds on these previous efforts and adds new features of 4D rendering (see, e.g., [20, 4, 14]) that have only recently become technically feasible to exploit interactively [11]. Such systems are valuable tools for mathematical research [16] as well as for volume and flow-field visualization applications [13, 15]. One of our goals is to develop techniques applicable to a real-time demonstration of 4D and volume-based 3D rendering applications in the CAVE [6]. In its present configuration, the CAVE is a Silicon Graphics Onyx/4 RealityEngine^2 with multiple graphics channels driving projectors for two wall displays and the floor, with simulator support for workstation code development.

2 Virtual Reality and Visualization of Geometry

In this paper we emphasize the visualization of challenging classes of mathematical objects through 3D volumetric rendering. This section outlines our viewpoints on the general issues involved in creating a virtual reality for such domains.

Mental models. Philosophers have long wrestled with the question of the nature of *reality*: we consider reality to be embodied in our personal *mental models* that derive from experience with natural phenomena and allow us to cope with the qualitative physics of everyday life. Virtual reality, then, is achievable in one of two ways: we may create simulated experiences that involve the subject by exploiting *existing* mental models and perceptual expectations; or we may attempt, by simulating phenomena with no real-world correspondence, to create *new classes* of mental models, thereby extending the human experience.

Necessary features. The basic features of the virtual reality systems that concern us here are:

1. **Immersion.** The system must physically involve the user by responding to viewer motions and actions, e.g., using a head-tracker, flying mouse, wand, etc. Regardless of whether the display medium is a simple through-the-window view or a CAVE, this intensifies the intuition-building experience.

2. **Interaction.** The virtual environment must respond to the participant's actions at a high frame rate and provide smooth and accurate tracking of the input devices to support realistic feedback to the viewer. The user must be able to make changes and observe immediate results in order to draw intuitive conclusions about the structure and behavior of the simulated environment.

3. **Visual realism.** Redundant *realistic* visual cues are needed to involve the participant, so we

should strive to include effects such as perspective, attenuation with distance, specular and diffuse shading, shadows, motion parallax, and occlusion. Providing such cues creates a more satisfying visual experience, in addition to providing qualitative intuitive information at a preconscious level [7].

Anticipating the future. One task of the virtual reality developer is to avoid the pitfalls of shortsightedness. In this respect, our philosophy is to extend our attention also to approaches that are not feasible in terms of current performance, but could be drastically accelerated in future hardware generations. Indeed, the development of appropriate algorithms to keep up with the rapid evolution of the hardware may be viewed as one of the most serious challenges we face: we may become imagination-limited long before the limits of the hardware technology are reached.

3 Interactive Realism

The goals of interactivity and realism are contradictory; we must apparently compromise between the speed of scan-conversion approximations and physically accurate but time-consuming ray-tracing methods. Here we present the fundamentals of a rendering semantics that has the potential to support an acceptable compromise.

3.1 The Toolbox

The following image-level abstractions form a set of fundamental tools in terms of which we can express a remarkable number of complex geometric rendering effects:

z-buffer. The z-buffer tests and optionally replaces a geometric value normally representing an object's distance seen through the pixel; complex effects can be achieved by selecting appropriate tests and replacement rules.

Frame buffer and accumulation buffer mathematics. Frame buffers support operations that act selectively on images and include addition, multiplication, logical operations, convolution, and histograms.

The accumulation buffer, which is separate from the frame buffer, is dedicated to image addition and scaling and usually has more bits per color component than the frame buffer. Typical applications involve averaging a number of images, as in Monte-Carlo integration.

Static and dynamic textures. A static texture is a common surface or volume texture map, while a dynamic texture is one that may vary between frames. For instance, we might simulate a window as a plane with the current outside view mapped onto it; as the scene outside changes, so does the texture map. In addition to storing surface color, texture maps may also serve as lookup tables for reflectance or shading functions.

Automatic texture vertex generation. Given a texture map containing a lookup table for a function, we may not know in advance what portion of the texture map is required. Texture vertices can be generated automatically by supplying a (possibly projective) transformation function from the geometry to the texture space. For example, given a volumetric woodgrain texture, the system automatically chooses the correct woodgrain to map onto a slice through the virtual block. For a more complete discussion of texture mapping operations, see [8].

3.2 First-order Effects

We consider a first-order effect to be one that requires a fixed, environment-independent number of primitive operations per scene element. In this section, we describe a set of useful first-order techniques that can be used to approximate rendering effects.

Planar reflection. The reflections from a flat surface can be defined by rendering the environment from a virtual viewpoint placed on the opposite side of the reflecting surface; this image is then mapped onto the surface, giving the impression of a mirror.

Non-planar reflection. An environment map is defined as the image of a perfectly reflecting sphere located in the environment when the viewer is infinitely far from the sphere. Given the view direction and normal at each vertex of a polygon, we can use automatic texture vertex generation to choose texture coordinates; this gives an approximation of infinite focal length reflection [8].

Shading maps. Texture maps can also be used as repositories for pre-computed reflectance functions (e.g., diffuse, Phong, Cook-Torrance, etc.). This method produces *much* better behavior than hardware lighting (i.e., Gouraud shading), which defines colors only at the vertices, and so cannot place specularities inside a large polygon.

Physically based luminaires. A diffuse emitter can be approximated by placing a projection point behind the planar emitter and using projective textures to shine a pre-computed cosine distribution texture map into the environment. The diffuse light thus emitted is multiplied by each surface's diffuse light color coefficient. Other distributions can be approximated by projecting different lookup tables. If we use shading maps to approximate the cosine term and distance attenuation at target polygons, we can approximate physical luminaires.

Shadows. Areas lit by a particular light can be defined as areas that that light "sees"; i.e., for sharp-edged shadows, we test whether a particular point can describe an unobstructed path to the luminaire. Using the z-buffer, we can define a depth texture map from the light's point of view. By projecting this texture map into the environment and z-buffering from the eye's point of view, we can compare the distance-to-the-light values of visible surfaces to the projected nearest-to-the-light value. If the eye sees a surface whose distance to the light is larger than the indicated minimum, it must be in shadow; if not, it is lit. If this function is applied over all pixels to define a binary black-white image, this can be multiplied by an image containing a shadowless lit image to construct a final image including appropriate shadows [18, 19]

3.3 Second-order Effects

Second-order effects provide more sophisticated image features, and involve iteration or multiple samples to generate a single image. These methods are sufficiently expensive, at present, to preclude their use in most interactive applications. However, these methods greatly increase visual satisfaction and hardware improvements will make them increasingly practical.

Multiple samples. The accumulation buffer allows an elegant implementation of Monte-Carlo methods over entire images. Thus, dynamic images such as shadow maps or reflection images can be defined by probabilistic sampling, producing smooth shadows or blurred reflections. Psychological research indicates that smooth shadows, in particular, are important for visual realism [7].

Iterated diffuse and specular reflection. If we approximate diffuse emission from a single polygon and produce an image of the incident light at another, we can then emit some of this light back into the environment. When this process is iterated, it becomes an approximation to radiosity and global illumination. This method has the advantage of being much faster than similar precomputations and has lower algorithmic complexity while maintaining important visual features.

Participating media. Approximate volume images may be produced by cutting multiple additive slices through the viewed space. By projecting lighting distributions and shadows onto these slices, we can approximate the scattering of light as it passes through a foggy environment, producing visible beams and similar volume-rendering effects.

3.4 Hardware Support

At present, only the simplest of the above techniques are viable in a software-only interactive system with a complex environment. As our needs for realism increase, so do our needs for graphics hardware support. For example, if we require Phong shading, but must implement it in software, the complexity of environments with which we can interact will be severely limited. However, given hardware texture-mapping support, we can precompute a lookup table to support an implementation of specular reflectance functions; the hardware can wrap this texture around the objects, interpolating between computed points to produce correct images of specular objects.

Our virtual geometry applications are designed to take full advantage of the hardware support of the Silicon Graphics Onyx RealityEngine^2. Its support for high-speed texture mapping, in particular, enables us to map large portions of our graphics computations directly onto hardware-supported primitives. For example, dynamic texture mapping and automatic texture vertex generation allow us to interactively simulate bizarre physical illumination models such as 4D light. Effectively, we have *transformed* the mathematical rendering model into a form expressible in terms of our hardware-supported high-speed toolbox. The exploitation of such transformations can greatly enhance user comprehension through improved feedback and perceived visual realism.

4 Visualization Effects Design

The particular system that we have implemented to explore virtual reality paradigms focuses on mathematical visualization in higher dimensions [11]. We create the illusion that the participant is immersed in the 3D, volumetric retina of a 4D cyclops interacting with objects in a 4D world. Among the issues we must address to achieve this are the following:

4D Depth. Perceiving 4D depth requires binocular fusion of a pair of distinct 3D volume images; a 3D human would effectively need 2 pairs of 3D eyes to see these images at the same time (and could not fuse them anyway). Thus we need to obtain 4D depth cues from other sources such as motion, occlusion, or depth color codes. For example, occlusions of surfaces by surfaces can be emphasized for visibility by painting or cutting away the more distant of two surfaces around an illusory intersection in a particular projection.

4D Motion Cues. Motion is an important factor in our ability to perceive 3D structure monocularly; either constant-angular-velocity rigid 3D rotation or periodic rocking is an adequate substitute for a stereo display. We use rigid 4D rotations to generate motion cues for our 4D monocular world that resemble familiar 3D motion cues.

3D Depth. Typical objects that we project out of 4D produce volume images, though our rendered images of thickened surfaces are simplified since we use a thin-surface approximation to achieve acceptable rendering performance. Thus seeing the 3D opaque exterior of our objects is not enough — we want to see *inside* the projected shape. This causes a problem: if we make a surface transparent but featureless, like a highly inflated balloon, there are not enough distinct features in the image to activate 3D stereo perception except along the outer edges of the object. Similarly, for volumetric objects whose interior is made of smooth internal "jellylike" solids, it is difficult to produce a strong impression of what may be a very complex internal 3D structure.

Texture. For human binocular vision to perceive a full 3D structure in one glance, smooth rendering methods are often deficient; they do not generate the image gradients necessary for the edge matching process used in stereo depth reconstruction. One technique to circumvent this problem is to spice up the featureless jelly with surface or volumetric textures. Such textures, which can be as simple as a set of grid lines or a regular or random lattice of points, provide a richer collection of image gradients to drive the stereographic matching process.

No Slices. A common approach to representing 4D objects is to slice them up and consider them as a sequence of 3D objects, often presented as a time-sequenced animation. We insist on holistic images for our imaginary 4D retina; humans are not adept at perceiving 3D objects from a time sequence of 2D slices, so we do not expect that 3D slices of 4D objects will be any easier.

Lighting. In everyday life, we are able to perceive 3D shapes in static photographs and drawings. We make certain assumptions about the nature of the objects and lighting conditions, and apparently infer the 3D structure using what is known in computer vision as a "shape from shading" algorithm. We see objects whose structure is revealed by the intensity gradations reflected from the object and by its cast shadows. Diffuse and specular highlights reveal *additional* information about the directions of surface or volume patches that is more specific than, for example, gradient or isosurface information. 4D lighting permits a similar holistic depiction of a 4D object, and the structure of the lighting in the 3D projection contains many subtle clues about the 4D structure and its orientation relative to the 4D lights and camera.

Shadows. To enhance the scene perception experience, we can provide auxiliary cues such as 3D shadows to supplement 3D stereo perception. One can also generate 4D shadow volumes to help reveal hidden 4D structure [12].

Occlusion. We exploit occlusion information to infer structure in 2D drawings representing the 3D world; a typical mathematical application would be the "crossing diagram" showing the unique 3D structure of a knotted loop of string using over/under crossing markings on a 2D diagram alone. Similar phenomena occur in 4D; non-intersecting surfaces in 4D may appear to intersect in a curve when projected to 3D, just as 3D lines may appear falsely to intersect when projected to 2D; pieces of 4D volumes may completely block out other 4D volumes in the 3D projection, just as 3D surfaces block (occlude) one another in 2D imaging. Single convex 3D and 4D objects have no occlusions, and so can be easily rendered using back-face culling. For 4D multiple-object scenes and non-convex objects, we can provide occlusion handling, crossing markings or depth-cued colors to emphasize the occurrence of 4D occlusion. This is not always possible to achieve interactively, since processing occlusions may be a memory-intensive or combinatorially explosive process.

Depth Coding. A number of techniques have been proposed to provide a sense of depth in 4D graphics (see, e.g., [2]); these include pseudocolor coding of depth, application of depth-dependent static or moving textures, and 4D-depth dependent opacity or blurring.

Redundancy. Typical 3D terrain maps and graphs have *redundant* coding of properties such as elevation. Pseudocolor, isolevel contours, ruled surface markings, and oblique views exhibiting occlusion, illumination effects, and shadows may all be combined in a single representation. 4D data representations also profit from such redundancy, so we add multiple 4D cues when possible.

In summary, the family of visual effects that we wish to achieve involves a wide variety of issues and representation technologies. The common thread is this: we examine holistic perceptual processes such as lighting and motion that serve us well in dealing with our 3D world, and exploit the 4D analogs of those processes to encode relevant information in the 3D volume image perceived by our hypothetical 4D being.

5 Interactive Interface Design

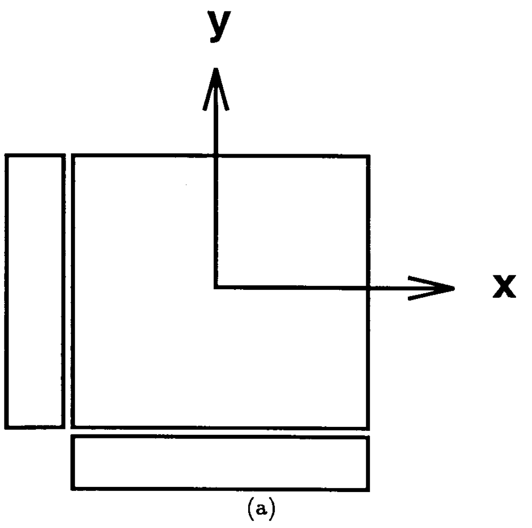
Our philosophy of 4D interaction is based on several fundamental assumptions about how human beings learn about the 3D world. We are all familiar with the fact that if we are driven around a strange town, we are much less able to find our way later than if we do the driving ourselves. Thus we seek 4D interaction modes that emphasize the involvement of the user and promote the feeling of direct manipulation, as though 4D objects were responding in some physical way to the motions of our input devices. Successful strategies should therefore significantly reduce the required user training time by exploiting analogs of familiar 3D direct manipulation.

Restricting ourselves for now to single compact objects lying in the center of our perceived CAVE space, we need several basic types of control: (1) the user moving around the 3D projected object itself; (2) rigid 3D motions of the object; (3) rigid 4D rotations (and perhaps translations) of the object; (4) 4D control of the orientation of the light ray (or rays) used in the shading and shadowing processes. The first two capabilities are standard for almost all CAVE applications.

The two 4D rotation tasks, however, require the following application-specific design considerations:

4D Orientation Control. Direct manipulation of 3D orientation using a 2D mouse is typically handled using a rolling ball [9] or virtual sphere [5] method to give the user a feeling of physical control. Figure 1a shows the effect of horizontal and vertical 3D rolling ball motions on a cube: supposing that the cube initially shows only one face perpendicular to the viewer's line of sight, moving the mouse in the positive x direction exposes an oblique sliver of the left-hand face, while motion in the y direction exposes the bottom face; reversing directions exposes the opposite faces. Long or repeated motions in the same direction bring cycles of 4 faces towards the viewer in turn. In the rolling ball method, circular mouse motions counter-rotate the cube about the viewing axis; in the virtual sphere method, the mouse acts as if glued to a glass sphere, so that at a certain radius along the x-axis from the center, a vertical mouse motion causes spinning about the viewing axis.

The extension of this approach to 4D is outlined in the appendix and described in more detail in [10]. Figure 1b is the 4D analog of Figure 1a: beginning with a fully visible transparent (volume-rendered) cube, which represents a single hyperface of a hypercube perpendicular to the 4D viewing vector, we move the 3D mouse along the x-axis to expose a volumetric sliver on the left; this is the oblique view of the left hyperface. Moving the 3D mouse along the y-axis exposes a volumetric sliver on the bottom; moving the 3D mouse along the z-axis exposes a volumetric sliver on the *back* of the original volumetric cube. Reversing directions brings up the opposite hyperfaces; long motions reveal cycles of 4 hyperfaces, but now there are *three* cycles, one each in the x, y, and z directions. How do we get ordinary rotations, say in the x-y plane? Moving



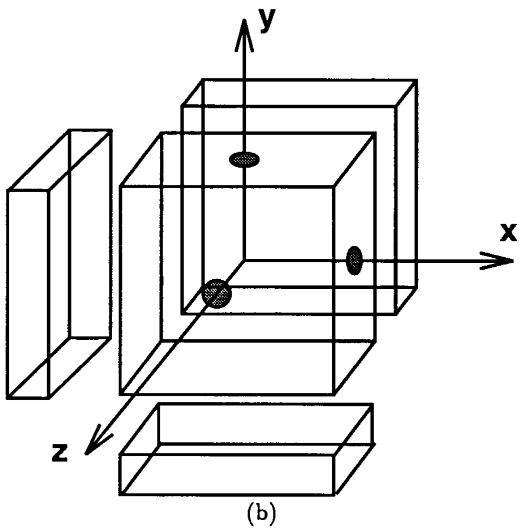


Figure 1: Schematic diagram comparing (a) 2D mouse control of a 3D object, and (b) 3D flying mouse control of a 4D object.

the 3D mouse in small circles in any plane produces counter-rotations of that plane, thus giving 3 more degrees of freedom, exhausting the 6 degrees of orientational freedom in 4D. The 4D virtual sphere action follows by exact analogy to the 3D case.

4D Light Control. Figure 2a shows a schematic diagram of a method for controlling the 3D lighting vector using a 2D mouse: the unit vector in 3D has only two degrees of freedom, so that picking a point within a unit circle determines the direction uniquely (up to the sign of its view-direction component). With a convention for distinguishing vectors with positive or negative view-direction components (e.g., solid or gray), we can uniquely choose and represent the 3D direction. Control of the vector is straightforward using the rolling ball: the lighting vector initially points straight out of the screen (up in the oblique view of Figure 2b), and moving the mouse in the desired direction tilts the vector to its new orientation, whose projection to the plane of Figure 2a is shown in the gray ellipse in Figure 2b. Rotating past 90 degrees moves the vector so its view-direction component is into the screen.

The analogous control system for 4D lighting, shown in Figure 2c, is based on a similar observation: since the 4D normal vector has only 3 independent degrees of freedom, choosing an *interior point* inside a solid sphere determines the vector uniquely up to the sign of its component in the unseen 4th dimension (the "4D view-direction component"). The rest of the control proceeds analogously. Since we cannot easily interpret 4D oblique views, we do not attempt to draw the 4D analog of Figure 2b.

6 Examples

A classic example of a non-trivial surface embedded in 4D is a knotted sphere, and this is the central demonstration we have implemented for the CAVE; Figure 3 shows the spun trefoil, the 4D knotted sphere closest in spirit to an ordinary 3D trefoil knot, while Figure 4 shows the twist-spun trefoil, which, astonishingly, can be shown to be unknotted in 4D.

The real-time display of these images is made possible by replacing the techniques of [14], which required up to half an hour per frame to render, with a dynamic texture map implementation of [11], resulting in update rates of up to 30 frames/second. The texture mapping support of the RealityEngine permitted us to represent the 4D lighting distributions on the surface using a dynamic texture map; the resulting transparent volumetric image was rendered using frame buffer addition with multiplicative opacity.

Occlusion computations remain too expensive for real time, and so we precompute the occlusions for one particular viewpoint and fix them to the object; this has the curious advantage that, when the object is rotated in 4D, one can see the explicit separation of the apparent self-intersections, and convince oneself that the "side view" shows that no self-intersections exist.

In Figure 5 we show a closeup of the 4D control feedback display, which reads out the current 4D light position and 4D orientation of the central knotted sphere. Figure 6 is a true volume-rendered object, the hypersphere, projected from 4D to show a 3D view of its "northern hemisphere;" grid lines and a volumetric speckle texture are added within the featureless volume of this object to give a clear stereographic image, as noted in Section 4.

Finally, in Figure 7, we step back to show how

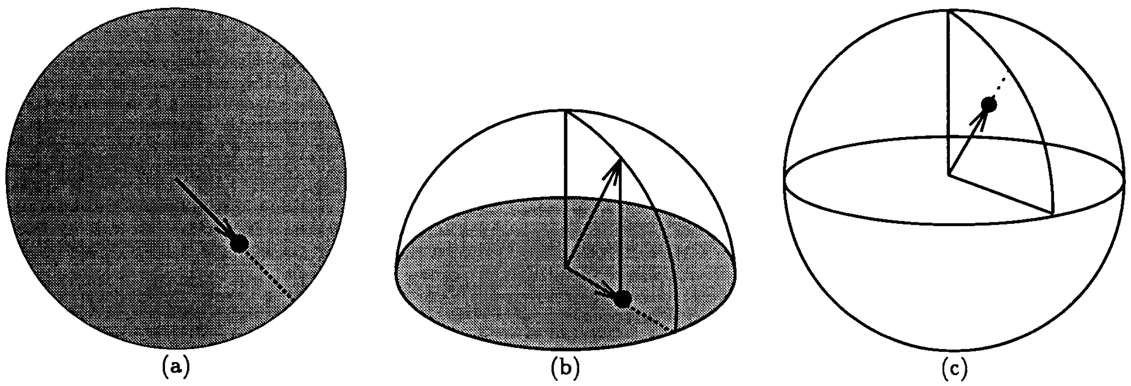


Figure 2: Schematic diagram comparing (a) selecting 2D point in disk to specify 3D light direction, shown obliquely in (b), and (c) 3D flying mouse control of a 4D light direction by picking 3D point inside solid sphere.

our mathematical world appears in a rich virtual workspace. To illustrate some of our other capabilities for general virtual geometry, we show in Figure 8 how the knotted sphere appears in a room illuminated by a light shining through foggy air. Note the satisfying effect of the 3D shadows cast in the room by the mathematical objects.

7 Conclusions and Future Work

We have described a wide spectrum of issues involved in the development of an ambitious virtual reality system that attempts to immerse the viewer in a technically correct four-dimensional world. The techniques required for this system include the optimization of rendering approximations through the use of hardware graphics operations, as well as task-specific approaches to enhancing user interaction. The optimizations used in this system apply also to general virtual reality performance problems.

While adapting our software to the CAVE, we faced parallelization issues that did not arise on a single screen. In addition to parallelizing the mathematics (e.g., multi-threading the projection of four dimensions to three), we dealt with other problems involving shared memory and resources. For example, one *must* avoid collisions on the graphics hardware, particularly during geometry transformations.

In its present form, this project is approaching the limits of the target graphics hardware. We now face the standard bottleneck of textured polygon fill rate, for instance. Extending the features of the system will require computations for which the RealityEngine hardware has no particular advantage; for example, the 4D occlusion calculation is still too computationally expensive for adequate interactive performance, as is depth-ordered transparency. However, the addition of resources such as hardware support for large dynamic 3D textures would enable us to attack even more challenging problems in virtual geometry.

Acknowledgments

This work was supported in part by NSF grant IRI-91-06389. We thank George Francis, Chris Hartman, and the NCSA CAVE personnel, as well as the members of the Electronic Visualization Laboratory at the University of Illinois/Chicago for their support.

References

[1] Banchoff, T. F. Visualizing two-dimensional phenomena in four-dimensional space: A computer graphics approach. In *Statistical Image Processing and Computer Graphics*, E. Wegman and D. Priest, Eds. Marcel Dekker, Inc., New York, 1986, pp. 187–202.

[2] Banchoff, T. F. Beyond the third dimension: Geometry, computer graphics, and higher dimensions. *Scientific American Library* (1990).

[3] Banks, D. Interactive manipulation and display of two-dimensional surfaces in four-dimensional space. In *Computer Graphics (1992 Symposium on Interactive 3D Graphics)* (March 1992), D. Zeltzer, Ed., vol. 25, pp. 197–207.

[4] Carey, S. A., Burton, R. P., and Campbell, D. M. Shades of a higher dimension. *Computer Graphics World* (October 1987), 93–94.

[5] Chen, M., Mountford, S. J., and Sellen, A. A study in interactive 3-d rotation using 2-d control devices. In *Computer Graphics* (1988), vol. 22, pp. 121–130. Proceedings of SIGGRAPH 1988.

[6] Cruz-Neira, C., Sandin, D. J., and DeFanti, T. A. Surround-screen projection-based virtual reality: The design and implementation of the CAVE. In *Computer Graphics (SIGGRAPH '93 Proceedings)* (Aug. 1993), J. T. Kajiya, Ed., vol. 27, pp. 135–142.

[7] Goldstein, E. B. *Sensation and Perception.* Wadsworth Publishing Company, 1980.

[8] Haeberli, P., and Segal, M. Texture mapping as a fundamental drawing primitive. In

Fourth EUROGRAPHICS Workshop on Rendering (June 1993), M. Cohen, C. Puech, and F. Sillion, Eds., pp. 259–266.

[9] HANSON, A. J. The rolling ball. In *Graphics Gems III*, D. Kirk, Ed. Academic Press, San Diego, CA, 1992, pp. 51–60.

[10] HANSON, A. J. Rotations for n-dimensional graphics. Tech. Rep. 406, Indiana University Computer Science Department, 1994.

[11] HANSON, A. J., AND CROSS, R. A. Interactive visualization methods for four dimensions. In *Proceedings of Visualization '93* (1993), IEEE Computer Society Press, pp. 196–203.

[12] HANSON, A. J., AND HENG, P. A. Visualizing the fourth dimension using geometry and light. In *Proceedings of Visualization '91* (1991), IEEE Computer Society Press, pp. 321–328.

[13] HANSON, A. J., AND HENG, P. A. Four-dimensional views of 3d scalar fields. In *Proceedings of Visualization '92* (1992), IEEE Computer Society Press, pp. 84–91.

[14] HANSON, A. J., AND HENG, P. A. Illuminating the fourth dimension. *Computer Graphics and Applications 12*, 4 (July 1992), 54–62.

[15] HANSON, A. J., AND MA, H. Visualization flow with quaternion frames. In *Proceedings of Visualization '94* (1994), IEEE Computer Society Press. In these Proceedings.

[16] HANSON, A. J., MUNZNER, T., AND FRANCIS, G. K. Interactive methods for visualizable geometry. *IEEE Computer 27*, 7 (July 1994), 73–83.

[17] PHILLIPS, M., LEVY, S., AND MUNZNER, T. Geomview: An interactive geometry viewer. *Notices of the Amer. Math. Society 40*, 8 (October 1993), 985–988. Available by anonymous ftp from geom.umn.edu, The Geometry Center, Minneapolis MN.

[18] REEVES, W. T., SALESIN, D. H., AND COOK, R. L. Rendering antialiased shadows with depth maps. In *Computer Graphics (SIGGRAPH '87 Proceedings)* (July 1987), M. C. Stone, Ed., vol. 21, pp. 283–291.

[19] SEGAL, M., KOROBKIN, C., VAN WIDENFELT, R., FORAN, J., AND HAEBERLI, P. E. Fast shadows and lighting effects using texture mapping. In *Computer Graphics (SIGGRAPH '92 Proceedings)* (July 1992), E. E. Catmull, Ed., vol. 26, pp. 249–252.

[20] STEINER, K. V., AND BURTON, R. P. Hidden volumes: The 4th dimension. *Computer Graphics World* (February 1987), 71–74.

A 4D Rolling Ball Formula

For completeness, we list the 4D rolling ball formula derived in [10] that is the basis for most of our 4D controls; this natural algorithm for 4D orientation control requires exactly three control parameters, thus making it ideally suited to the "flying mouse" or CAVE "wand" 3-degree-of-freedom user interface devices. Let $\vec{X} = (X, Y, Z)$ be a displacement obtained from the 3-degree-of-freedom input device, and define $r^2 = X^2 + Y^2 + Z^2$. Take a constant R with units 10 or 20 times larger than the average value of r, compute $D^2 = R^2 + r^2$, compute the fundamental rotation coefficients $c = \cos\theta = R/D$, $s = \sin\theta = r/D$, and then take $x = X/r, y = Y/r, z = Z/r$, so $x^2 + y^2 + z^2 = 1$. Finally, rotate each 4-vector by the following matrix before reprojecting to the 3D volume image:

$$\begin{bmatrix} 1 - x^2(1-c) & -(1-c)xy & -(1-c)xz & sx \\ -(1-c)xy & 1 - y^2(1-c) & -(1-c)yz & sy \\ -(1-c)xz & -(1-c)yz & 1 - z^2(1-c) & sz \\ -sx & -sy & -sz & c \end{bmatrix}$$

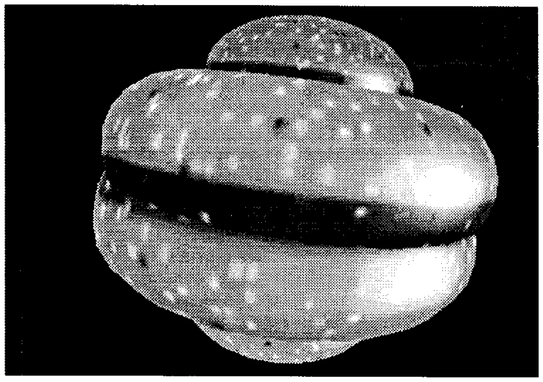


Figure 3: Closeup of spun trefoil knot in the CAVE simulator.

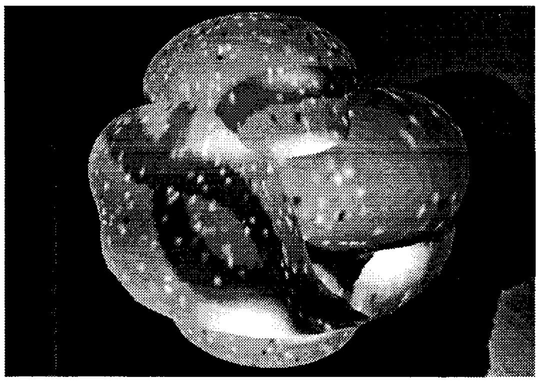


Figure 4: Closeup of twist–spun trefoil apparent knot in the CAVE simulator.

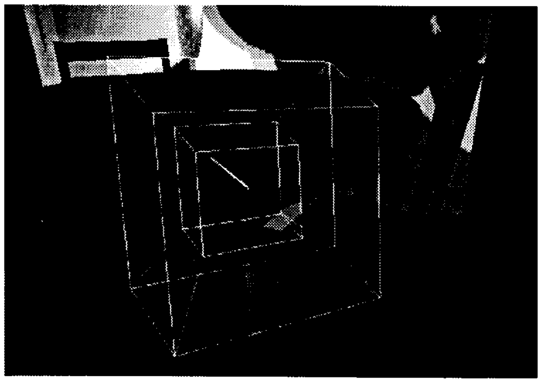


Figure 5: 4D control feedback display; single line shows light direction, wire–frame shows 4D orientation referred to a hypercube.

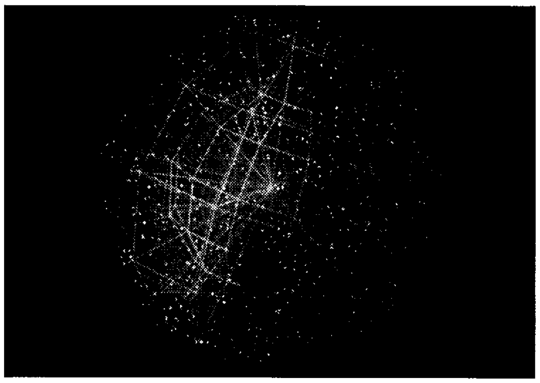


Figure 6: The "solid textured beach–ball" view of the hypersphere.

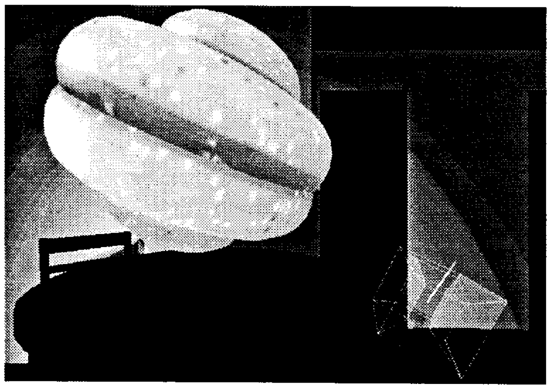


Figure 7: User view of knotted sphere and controls inside a rich virtual room.

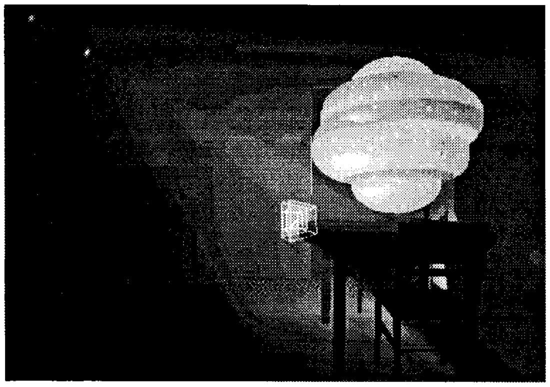


Figure 8: Adding 3D light and fog to the virtual workspace.

(See color plates, page CP-17.)