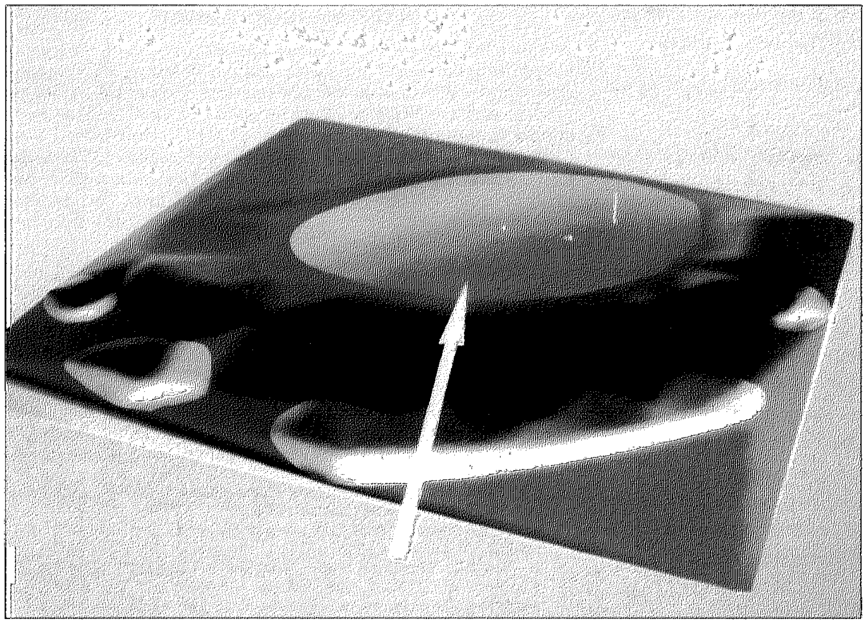


Figure 15. Wykagyl Country Club 16th hole.

derers, cameras, lights, actors, models, and splines. We also developed classes specifically for golf modeling and analysis. One set of analysis classes implements the initial- and boundary-value solution algorithms in *Numerical Recipes.*[7] User-interface classes for the HP widgets[12] permitted us to create a custom and portable X Window user interface.[13] The interpreted LYMB environment let us customize the user interface for easy operator interaction during the broadcast.

The Golf Green class reads survey data and creates a polygonal model of the green, perimeter, and skirt. It also responds to *@(x,y)z?*, returning the value of the elevation at the requested Cartesian location on the green surface. Another message, *@(x,y)normal?*, returns the surface normal at the point (x, y). The initial-value problem solver uses these messages. Golf Green has a scale factor that scales the elevations and normals so the shooting method can control the flatness of the green. We implemented the shooting method in a LYMB script that uses loops, the initial-value solver, and Golf Green classes.

The system runs on Sun 3/4, Hewlett-Packard 9000, Stardent GS2000, Digital Equipment DS5000, and Silicon Graphics 4D workstations. Rendering classes for vendor-specific hardware permit fast response on these systems. For instance, the Silicon Graphics workstation renders a typical green at 5 frames per second. Initial-value problems on this machine take less than a second. Boundary-value solution times depend on the green topography, coefficient of friction, and distance from the hole. For distances less than 10 feet, solution times are under 3 seconds. Distances longer than 30 feet require an average of about 20 seconds.

User interface

By integrating user-interface classes with existing applications, including the parser and animation classes, we created a powerful visualization environment. Figure 16 shows one user interface to the golf simulation. The green can be viewed from different points in space with a virtual camera controlled by a pointing device (a mouse). The user performs the rest of the interactions using various widgets in the control panel on the right:

- Pin & Ball pops up sliders to change the position of the pin and the ball on the green. Whenever the ball or pin position is changed, the system places the virtual camera in a natural player's position, so the ball and the hole are in the field of view. After the user changes the ball or hole position, three widgets immediately show suggested speed, direction, and distance from the ball to the hole.
- Arrow widgets change the initial putt speed and direction. Footprints and a putter head are displayed beside the ball each time the direction is changed. Zero direction corresponds to the straight line from the ball to the hole.
- The Putt selection shows the ball's trajectory on the green calculated on the basis of the defined initial speed and direction.

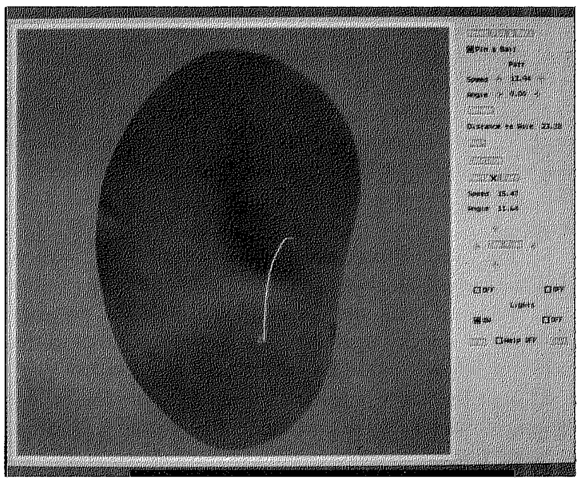


Figure 16. Golf simulation user interface.

Much of the mathematical literature focuses on surfaces (manifolds of dimension 2) embedded in four dimensions and ways of displaying their projections to three dimensions. A novel property of the 4D world not present in three dimensions is the existence of nontrivial volumetric objects (manifolds of dimension 3); accordingly, other work such as that of Steiner and Burton[9] and Carey, Burton, and Campbell[10] has addressed the problem of how to render 3-manifolds in a 4D world and show the results in an ordinary 2D image. The 4D rendering algorithms we describe in this article share many basic features with the algorithms presented by these authors. However, no previous 4D rendering method applies to points, curves, or surfaces in four dimensions. Our technique[11] generates images revealing new, intrinsically four-dimensional features of manifolds with dimension less than 3 embedded in a 4D world.

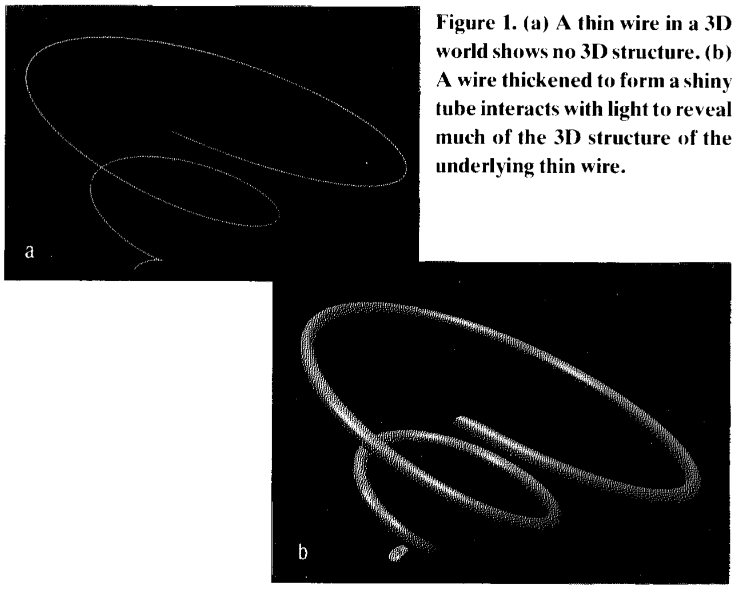


Figure 1. (a) A thin wire in a 3D world shows no 3D structure. (b) A wire thickened to form a shiny tube interacts with light to reveal much of the 3D structure of the underlying thin wire.

Approach

Our approach can best be understood by considering the problem of rendering a tangle of very thin wire in 3D space. At each point, the wire has one local tangent direction and a normal plane containing a set of normal directions parameterized by a circle. Light falling on the wire is not reflected toward the image plane because there is no flat mirrorlike surface to cause such a reflection. This is just another way of saying that points on the wire do not have one unique normal vector, as required by elementary models for computing shading from illumination, but have a *family* of normal directions lying in a plane. However, if we *thicken* the thin wire by adding a shiny circle in the normal plane at each point, it becomes a *tube* that reflects light beautifully and shows a great deal of 3D structure. Figure 1 shows the contrast between these two situations. The thickened wire also shows clearly where one part of the wire passes in front of another part, giving even more 3D information.

Since surfaces embedded in four dimensions present the same sort of rendering difficulties as wires embedded in three dimensions, we adopt a similar solution. The fundamental idea of our surface-rendering technique is to attach a shiny circle to each point of the surface to make it "show up" when we shine 4D light on it.

Rendering in *D* dimensions

A simple way of making sense of what it means to render a mathematical object that lives in a 4D world is to consider step by step how we might render objects into images in simpler worlds. Figure 2 illustrates graphically the process of rendering a scene in two, three, and four dimensions. In the 2D world of Figure 2a, light rays can reflect directly off curves to form an image in the pixels of the *view line*. Points are also interesting objects in two dimensions, but they do not have a single normal

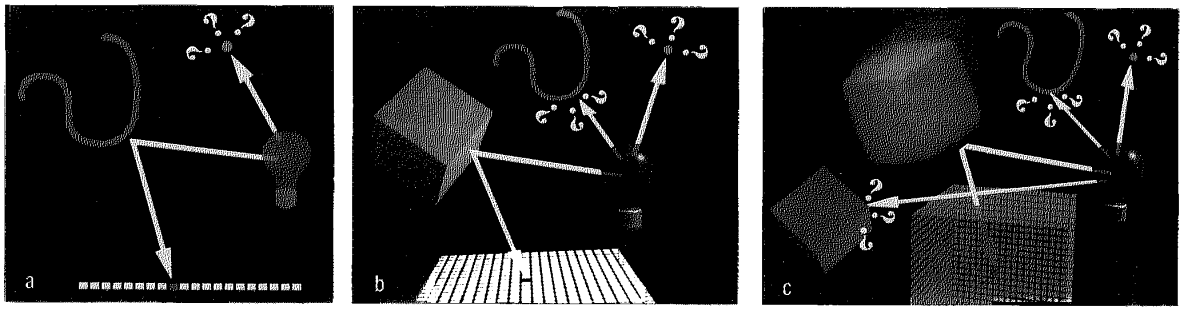


Figure 2. In each dimension *D*, only objects of dimension (*D*–1) reflect light in a unique direction to form an image in the (*D*–1)-dimensional viewing region. (a) Two dimensions, (b) three dimensions, (c) four dimensions.

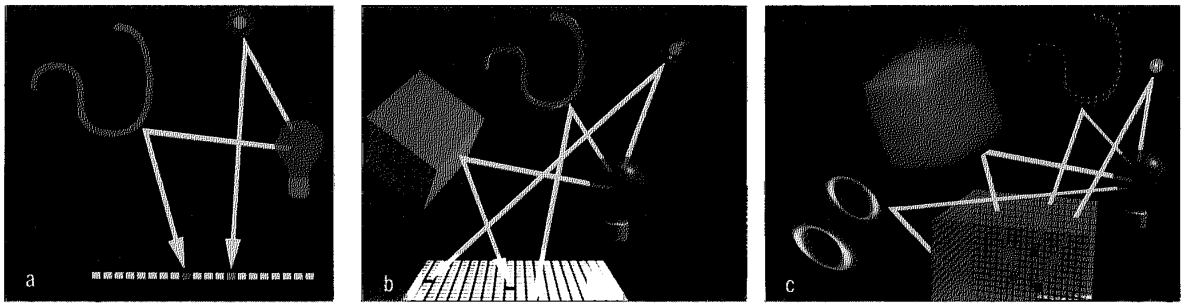


Figure 3. In each dimension D, we can make objects of any dimension appear to reflect light in a unique direction by thickening them in a radially symmetrical way so they acquire dimension (D–1). They then form proper images in the (D–1)-dimensional viewing region. (a) Two dimensions, (b) three dimensions, (c) four dimensions.

vector and light does not reflect coherently from them to form a useful shaded image. In the 3D world of Figure 2b, surfaces have unique normal vectors at each point. Lighted surfaces form images in the pixels of the *view plane.*[12] Bare points and lines in three dimensions do not reflect light to form good shaded images. Finally, as Figure 2c shows, an image in a 4D world is a volume that we call the *view volume*. Volumetric objects (3-manifolds) reflect light properly to form shaded images in the view volume, but points, curves, and surfaces do not.

Now suppose we want to "see" objects such as points in two dimensions, points and curves in three dimensions, and points, curves, and surfaces in four dimensions. We need to make these objects "thicker" so they have unique normal vectors; only then can we use conventional models to form their shaded images in the viewing region. The simplest approach is to expand each object in a radially symmetrical way, going out from each point. Thus, we make a point visible in two dimensions by adding a circle that turns it into a little curve. We expand points in three dimensions into spheres, and make a curve visible by sweeping it out with a circle to turn it into a cylindrical tube. In four dimensions, points turn into 3-spheres, curves become swept 2-spheres, and we make a surface visible by attaching a shiny circle at each point.

Table 1. To make an object of dimension d in a D-dimensional world renderable, give it dimension (D–1) by attaching the indicated (D–d–1)-dimensional object to each point.

	Add indicated structure to make object renderable			
World Dimension	Points $d = 0$	Curves $d = 1$	Surfaces $d = 2$	Volumes $d = 3$
$D = 2$	Circle	-		
$D = 3$	Sphere	Circle	-	
$D = 4$	3-Sphere	Sphere	Circle	-

Table 1 summarizes the procedure for modifying objects in each dimension to make them usable with conventional lighting models and shaded rendering methods in that dimension. Figure 3 illustrates the geometric modifications graphically for comparison with Figure 2.

We can summarize the procedure in a general rule:

> To make an object of dimension d (a d-manifold) interact sensibly with light rays in D dimensions, attach to each point of the object a radially symmetrical (spherical) manifold of dimension ($D - d - 1$) lying in the object's normal space.

The result is always a manifold of dimension ($D - 1$), the same dimension as the viewing region. For example, in two dimensions ($D = 2$), a point (which has dimension $d = 0$) must be expanded into a circle, which is technically a sphere of dimension 1. In four dimensions ($D = 4$), a line ($d = 1$) must have ordinary balloonlike 2D spheres attached to each point to expand it into a 3-manifold.

Provided we use the methods just described to convert all our objects into 3-manifolds in a 4D world (that is, objects defined by volumes with 4D vertex coordinates), we can create shaded images by analogy with 3D rendering techniques. We replace projection of visible 3D polygon faces into the view plane with projection of visible 4D volume elements into the view volume. The projected volume is then scan converted to the raster of view-volume voxels. Multiple volume elements might lie on the same 4D line of sight. We can account for the resulting occlusion effects using methods such as a "w-buffer," the 4D analog of a z-buffer, to eliminate parts of objects that are behind other objects relative to the 4D viewpoint. In the following section, we show how we can handle and exploit 4D shadows using similar analogies to 3D methods.

Projections and shadows

Here we outline the mathematical principles needed to compute projected images of objects and their shadows in D dimensions.

Projecting to a lower dimension

In three dimensions, both the view plane and the ideal walls on which shadows are cast are typically 2D rectangles. We argue that in four dimensions the view volume is a 3D rectangular solid and shadows are cast on "walls" that are rectangular solid subsets of hyperplanes. Objects appear in the view volume as 3D projections from four dimensions: that is, all points on the 4D object that lie on a ray from the 4D focal point through a 3D point in the view volume are projected to that single 3D point in the image. We must use depth-buffering methods to choose the opaque scene point nearest the focal point or to combine transparent objects. Furthermore, a darkened, smokelike cloud appears in the view volume whenever a ray cast from the 4D light source onto a visible "wall" is obstructed by an object to form a shadow.

To project from D dimensions to $(D-1)$-dimensional view hyperplanes or shadow hyperplanes, we let $\mathbf{X} = (X_1, X_2, \ldots, X_D)$ be the location of either a light source or a viewpoint, and let the equation satisfied by points $\mathbf{x}$ in the image (hyper)plane be

$$\hat{n} \cdot \mathbf{x} = c \qquad (1)$$

where $\hat{n} \cdot \hat{n} = 1$. Given any known point $\mathbf{H}$ in the (hyper)plane, we can determine the value of c using $c = \hat{n} \cdot \mathbf{H}$. We find the image of a scene point $\mathbf{P} = (P_1, P_2, \ldots, P_D)$ by substituting into Equation 1 the parametric equation of a line joining the viewpoint or light source $\mathbf{X}$ to the scene point, $\mathbf{x}(t) = \mathbf{X} + t(\mathbf{P} - \mathbf{X})$, and solving for $t_0 = (c - \hat{n} \cdot \mathbf{X}) / (\hat{n} \cdot (\mathbf{P} - \mathbf{X}))$. The image point $\mathbf{I}$ lying *within* the image (hyper)plane (Equation 1) is then

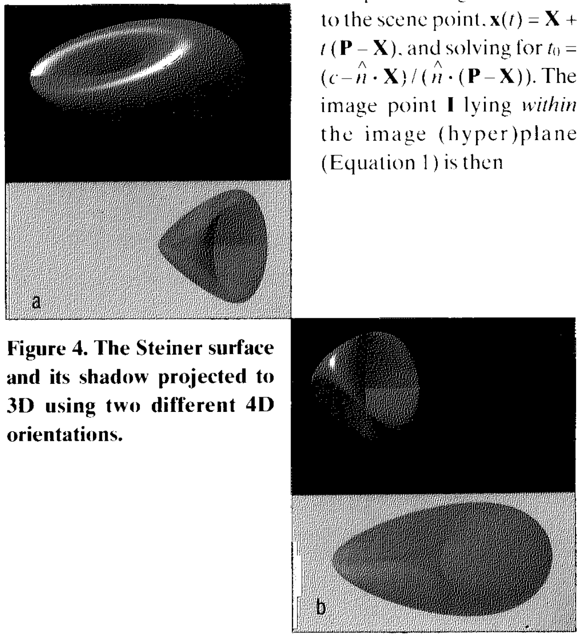


Figure 4. The Steiner surface and its shadow projected to 3D using two different 4D orientations.

$$\mathbf{I} = \mathbf{x}(t_0) = \frac{\mathbf{X}(\hat{n} \cdot \mathbf{P} - c) + \mathbf{P}(c - \hat{n} \cdot \mathbf{X})}{(\hat{n} \cdot \mathbf{P} - \hat{n} \cdot \mathbf{X})} \qquad (2)$$

The role of shadows

The information provided by a single shadow seen alongside the object in a single view is like information arising from a *second viewpoint* at the light source. Therefore, even a single shadow provides some stereographic depth information if we know the location of the light source and of the (hyper)plane on which the shadow is cast. Multiple shadows cast by multiple light sources can provide additional constraints resembling the various views in an engineering drawing (which would have four views in four dimensions).

Figures 4 and 5 show examples of (unthickened) 2D surfaces embedded in four dimensions along with their 4D shadows. We made these images by projecting the object and its shadow into three dimensions, illuminating the object's opaque projection with 3D lighting, treating the shadow as a gray semitransparent film, and rendering everything into a conventional 2D image. The shadows here are thin surfaces, artificially thickened for visibility, as we might treat the shadow of a wire in three dimensions. Rotating one of these objects in four dimensions interchanges its image outline with that of its shadow, as shown in the figures. We give the mathematical descriptions of these objects later.

Intensity shading methods

Images of illuminated 3D objects produce a wealth of orientation cues that the human visual system can interpret reliably, apparently by imposing constraints on the ambiguous data. Shaded images of 4D objects also produce rich orientation cues that are potentially useful, provided we can learn to interpret them. In this section, we discuss the issues involved in producing shaded images of objects in four dimensions.

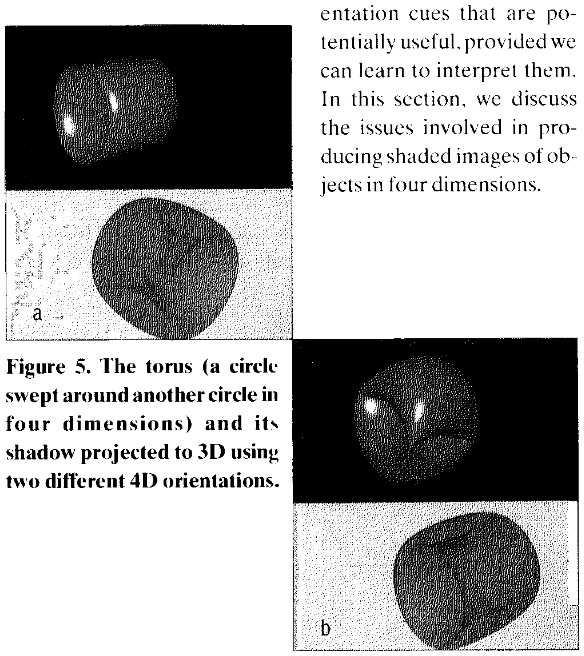


Figure 5. The torus (a circle swept around another circle in four dimensions) and its shadow projected to 3D using two different 4D orientations.

Extending rendering from 3D to 4D

The rendering process in four dimensions is closely analogous to that in three dimensions. In three dimensions, a typical Gouraud or Phong algorithm for rendering the surface of a solid object into the 2D view-plane raster follows these steps:

1. Divide planar faces into triangles.
2. Compute the normals at the face vertices. For example, take the gradient of an implicit surface defined by a single function of the 3D coordinates, compute the cross product of tangents to a parametric surface, or average the normals of the surrounding faces in the general case.
3. Project each triangle to the 2D view plane.
4. In the view plane, interpolate the intensities (Gouraud shading) or normals (Phong shading) from vertex to vertex to get the values at the beginning and end of each scan line.
5. Interpolate these values along the scan line to get the intensity or normal at an arbitrary pixel in the projected triangle.
6. For Phong shading, compute the intensity at the pixel using the interpolated normal.
7. Use a z-buffer or equivalent method to eliminate hidden surfaces if required.

In four dimensions, we alter these steps to render the volume element:

1. Subdivide the volume into tetrahedrons embedded in four dimensions. Take care to avoid subdivisions that leave empty volume gaps between adjacent nonplanar faces if a rectangular lattice is being used.
2. Compute the intensities or normals at the tetrahedral vertices. For example, find the normals by taking the gradient of a *volume* defined implicitly by a single equation in the four coordinate variables, by taking the cross product (see below) of the tangents, or by averaging the normals of the surrounding *volumes* in the general case.
3. Project each tetrahedron to the view volume.
4. In the view volume, interpolate the intensities (Gouraud shading) or normals (Phong shading) from vertex to vertex to get the values at the *vertices of each polygon* lying in the scan planes slicing up the tetrahedron.
5. In each scan plane, we now have a planar polygon representing a slice of the tetrahedron's voxels. Using the known values at each vertex of this polygon, interpolate to find the values at the polygon's internal voxels using the standard 3D view-plane interpolation.
6. For Phong shading, compute the intensity at each voxel using the interpolated normal.
7. To eliminate occluded volumes from the view volume, use a 4D depth buffer (w-buffer) or equivalent method and store only the intensity of the nearest object lying on a given ray into the view-volume voxels.

Smooth shading

In three dimensions, a typical shading algorithm computes the normal vector $\hat{n}$ at a surface point (usually a polygon vertex) and assigns a Lambertian diffuse intensity

$$I_D = I_0 \hat{n} \cdot \hat{L} \tag{3}$$

to the point, where $\hat{L}$ is the unit vector from the point to the light source and the dot product is taken as zero when negative (the normal is pointing away from the light). If both sides of a polygon might be visible, the absolute value may be used instead or the surface may be doubly covered (for example, for one-sided surfaces).

We can actually consider $\hat{n}$ to be proportional to the *cross product* of the ordered tangent vectors $\mathbf{P}(u, v)$, $\mathbf{Q}(u, v)$ at a local point (u, v) in the surface, so

$$I_D = \frac{I_0}{H} \operatorname{Det} \begin{vmatrix} L_1 & P_1 & Q_1 \\ L_2 & P_2 & Q_2 \\ L_3 & P_3 & Q_3 \end{vmatrix} \tag{4}$$

where $\mathbf{L}$ is the vector to the light source and we choose the normalization $H = \|\mathbf{L}\| \times \|\text{Cofactor } \mathbf{L}\|$ to make the maximum value of the determinant be unity.

In four dimensions, the shading equation for 3-manifolds is obtained either from Equation 3 with a four-vector normal or by expanding the determinant in Equation 4 to four columns and placing the tangent directions ($\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$) at a local point (u, v, θ) on the 3-manifold into the columns, yielding the form

$$I_D = \frac{I_0}{H} \operatorname{Det} \begin{vmatrix} L_1 & P_1 & Q_1 & R_1 \\ L_2 & P_2 & Q_2 & R_2 \\ L_3 & P_3 & Q_3 & R_3 \\ L_4 & P_4 & Q_4 & R_4 \end{vmatrix} \tag{5}$$

For points (dimension $d = 0$), curves ($d = 1$), and surfaces ($d = 2$), there are zero, one, or two available tangent vectors instead of the three needed in Equation 5, so we are missing critical information needed to compute a 4D intensity.

Specularity

Specular highlights are a dominant source of intuitive shape information in 3D renderings, so we expect specularity to be equally important in our perception of 4D geometry. We can construct the 4D analog of the specular shading contribution to an object's image in various ways.[12] Here we use a Phong specular shading approach computed by replacing the lighting vector $\hat{L}$ with the normalized sum $\hat{B}$ of the camera direction $\hat{C}$ and the lighting vector $\hat{L}$, and then raising the appropriate dot product or determinant to a high power k. Thus we simply add to the shading equation a specular term such as

$$I_S = I_1 |\hat{B} \cdot \hat{n}|^k \tag{6}$$

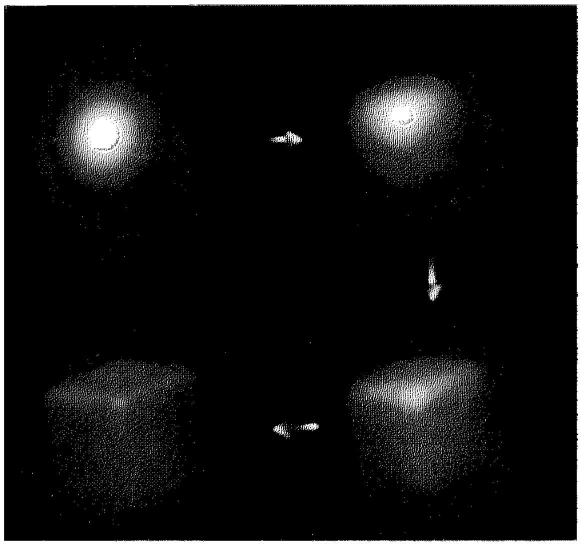


Figure 6. Four stages in the superquadric interpolation from a 3-sphere to a hypercube or tesseract.

where n is the appropriate 4D normal and the dot product is set to zero when negative. We then render into the view volume by summing both the diffuse and specular components of the intensity at each voxel.

Examples of rendered volumes

We can render volumes, or 3-manifolds, embedded in a 4D world straightforwardly using the shading methods presented above. Next we give some examples of such renderings to provide a context for the more complex thickened-surface renderings to follow.

The 3-sphere and the hypercube or tesseract are classic subjects of study in four dimensions. Here we examine them together using the fact that the *superquadric*[3]

$$|x|^{\gamma} + |y|^{\gamma} + |z|^{\gamma} + |w|^{\gamma} = r^{\gamma}$$

reduces to the 3-sphere when $\gamma = 2$, but asymptotically approaches a hypercube or tesseract as $\gamma \to \infty$. Figure 6 shows how the rendered object evolves in appearance as it makes the transition from a hypersphere to a hypercube. Figure 7 shows how the hypercube's specular shading changes as the viewpoint rotates in four dimensions around the scene center. Just as an edge-on face of a 3D cube reduces to a line when it is aligned with the viewing angle, the cubic volumes forming the bounding "hyperfaces" of the hypercube approach planar polygons embedded in the view volume when aligned with the 4D view direction.

Examples of surfaces in 4D

In this section we present our results for some classic examples of surfaces from 4D topology. Applications of the method to other situations, such as the rendering of points and curves embedded in four dimensions, have been published elsewhere.[11]

General approach

Surfaces are intrinsically well defined for shaded rendering in three dimensions, but not in four dimensions. For surfaces in four dimensions, we must identify the normal plane at each point and place a circle or ring with a small radius in this plane centered at the point. We can find a description of the ring attached at each point by taking the local tangent vectors $P^{\mu}(u, v)$ and $Q^{\mu}(u, v)$ at a point $x^{\mu}(u, v)$ of a surface parameterized by the two variables (u, v) and using a Gram-Schmidt procedure to find candidates for an orthogonal coordinate basis (N_1^{μ}, N_2^{μ}) of the plane perpendicular to the surface. That is, we require $\mathbf{N}_i \cdot \mathbf{P} = \mathbf{N}_i \cdot \mathbf{Q} = \mathbf{N}_1 \cdot \mathbf{N}_2 = 0$. Once we find $N_1^{\mu}(u, v)$ and $N_2^{\mu}(u, v)$, we normalize them and substitute the 4D normal vector

$$\hat{n}^{\mu}(u, v, \theta) = \hat{N}_1^{\mu}(u, v) \cos \theta + \hat{N}_2^{\mu}(u, v) \sin \theta \qquad \textbf{(7)}$$

into the 4D versions of Equations 3 and 6. The three variables parameterize the corresponding 3-manifold $X^{\mu}(u, v, \theta) = x^{\mu}(u, v) + r\hat{n}^{\mu}(u, v, \theta)$ for which $\hat{n}^{\mu}(u, v, \theta)$ is the current normal direction and r is the radius of the attached circle. One technical point is that we may have difficulty finding a smooth transition among coordinate patches for the 3-manifold over the entire surface. With one-sided surfaces, for example, we might need to cover the surface twice.

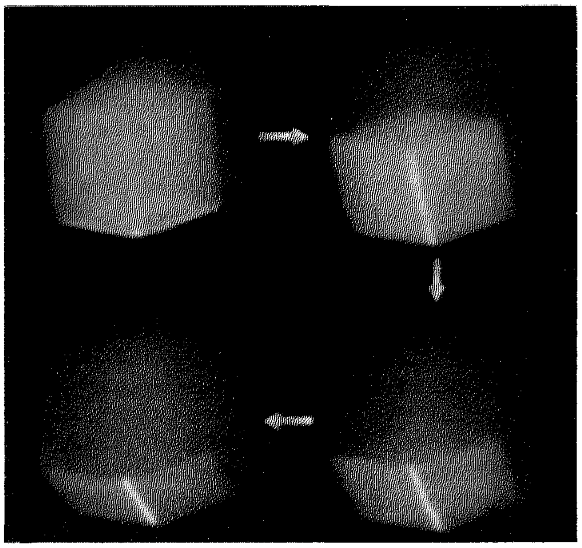


Figure 7. A succession of viewpoints of a 4D hypercube as the 4D viewpoint rotates around the center of the scene.

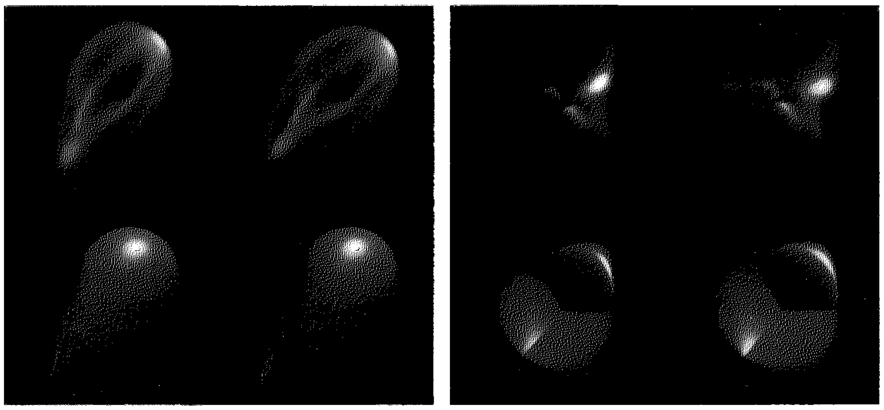


Figure 8. Cross-eyed stereo pairs of two views of a Steiner surface differing by their 4D orientation. Top: the thickened Steiner surface rendered in four dimensions with 4D lighting. Bottom: the standard Steiner surface projected to three dimensions and rendered using 3D methods. The images on the left side are for the right eye, and the images on the right side are for the left eye. To view in stereo, place a finger on the page between the two images and focus on the finger while moving it toward your nose until you see three distinct images. The center image should appear in three dimensions.

Steiner surface

Hilbert and Cohn-Vossen[14] gave a set of equations for a nonsingular embedding of the projective plane in four dimensions known as a Steiner surface. We represent this surface parametrically by the equations

$$x = \cos^2 u \cos^2 v - \sin^2 u \cos^2 v$$
$$y = \sin u \cos u \cos^2 v$$
$$z = \cos u \sin v \cos v$$
$$w = \sin u \sin v \cos v$$

where $0 \le u < \pi$, $0 \le v < \pi$. We compute the tangent vectors locally by taking partial derivatives with respect to u and v and orthonormalizing as required.

When we rotate this equation in four dimensions, it changes smoothly from the classic "cross cap" form of the projective plane to Steiner's Roman surface, depending on the rotation axes we choose. Figure 4 shows views of this surface from different 4D viewpoints produced by projecting the surface and its shadow to three dimensions before rendering using conventional 3D lighting methods. The use of shadows greatly clarifies the nature of the projection from four dimensions to three dimensions. If we animate this rotation, we can simultaneously see the two states of the shape and watch them exchange places as the rotation proceeds.

Figure 8 shows stereo pairs of the shaded volume rendering of the Steiner surface. We thicken each point by placing a small specular circle in the normal plane and illuminate the resulting 3-manifold by a single 4D point light source. To distinguish 4D from 3D effects, we render the full 4D thickened object along with the 3D rendering of its projection. We give images for two separate 4D rotation angles, showing the dramatic changes in the apparent shape and its 4D specular reflections as the orientation evolves.

4D torus

Next we look at the torus, another 4D surface described by Hilbert and Cohn-Vossen.[14] The familiar torus in three dimensions is strongly curved; the 4D version is intrinsically flat, like the surface of a cylinder in three dimensions. We formulate the equations in four dimensions simply by taking two separate circles, one in the first two coordinates and the other in the second two coordinates of the 4D space:

$$x = \cos u$$
$$y = \sin u$$
$$z = \cos v$$
$$w = \sin v$$

Figure 5 shows the 3D projection of the torus object (with 3D rendering) and its shadow for two viewpoints in a 4D rotation sequence. Again, an animation of this rotation shows how the object and its shadow continually exchange appearances. Using only the 3D projection, it is very difficult to grasp the flat nature of the intrinsic surface in four dimensions.

Figure 9 shows stereo pairs of the torus rendered in four dimensions with 4D lighting and in two different 4D orientations. The 3D projection with 3D lighting is shown beneath for comparison. Here the 4D specularities again give a dramatically different picture and provide some insight into the flatness of the surface in four dimensions.

Knotted sphere

Finally, we examine the 4D version of a knot. Knotted curves can exist only in three dimensions, since in four dimensions a knot can always be undone by jumping out in the fourth dimension. In four dimensions, however, we can have two *surfaces* that clash at a point and cannot pass by one another. Thus, in

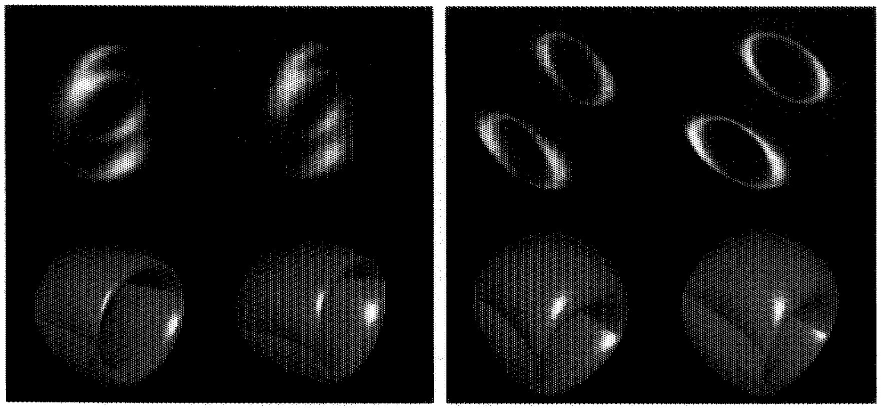


Figure 9. Cross-eyed stereo pairs of two 4D orientations of the torus rendered both in four dimensions and three dimensions.

four dimensions, knots are formed by closed surfaces such as the sphere, rather than by closed curves.[7]

We construct a knotted sphere by taking a freehand curve of a single overhand knot and rotating that knot into the fourth dimension about an axis containing both ends. After one full rotation, the knot rejoins itself and forms a closed surface. Any slice through this surface by a hyperplane in four dimensions gives a knotted curve in three dimensions. Figure 10 presents a stereo rendering of this knotted sphere in four dimensions, with the 3D projection of the surface below for comparison.

Conclusion

We have proposed a family of techniques for creating intuitively informative shaded images of 4D mathematical objects. We deliberately concentrated on depiction methods close to familiar 3D shaded images, as opposed to range images and slices, which have an artificial appearance.

If we consider the requirements for a robust 4D visualization system, our work raises a number of unanswered questions. In particular, we have not proved that the shaded images we produce are *interpretable* by human observers. We do not know how the necessary interpretation techniques can be taught to human users, nor do we know that the desired interpretation of the images is *cognitively feasible* within the limitations of the human intellect.

A robust approach to visualization should address such issues, leading us to propose a general visualization paradigm[11] for applications of the sort described here:

> A useful data depiction must allow the viewer to reconstruct a consistent model of the original data.

In our case we can meet this requirement in principle by extending to four dimensions the "shape-from-shading" techniques of 3D machine vision described by Horn,[15] much as we extended 3D rendering methods to four dimensions. That is, we can argue that *some intellect*, possibly superhuman, can in fact understand the 4D images we produce.

Further study of the utility of 4D shading for understanding the fourth dimension must address these deep questions. We hope the methods presented here will provide a new fundamental tool for the fascinating field of 4D visualization. ❑

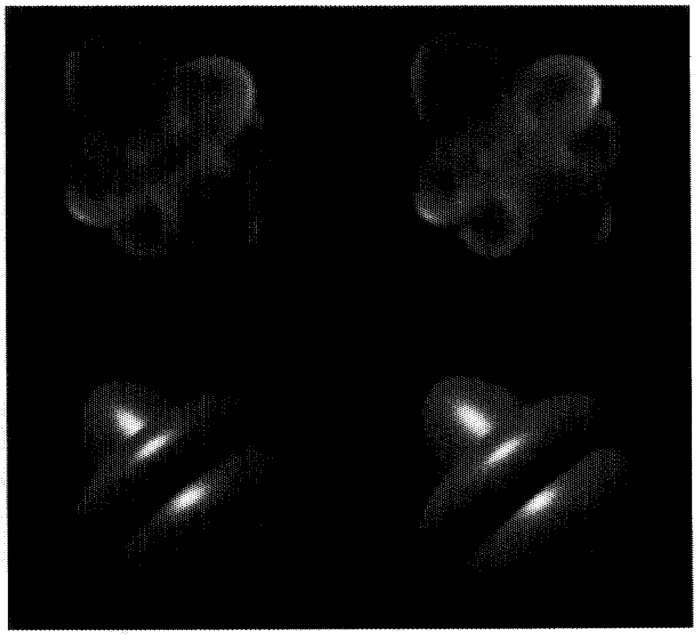


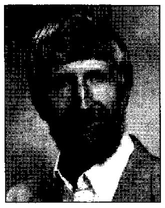
Figure 10. Cross-eyed stereo view of the knotted 2-sphere rendered in four dimensions with 4D lighting and also rendered in its 3D projection with 3D lighting.

Acknowledgments

We thank Brian Kaplan for his assistance with Figures 2 and 3, and gratefully acknowledge the role of CICA, the Indiana University Center for Innovative Computing Applications, in supporting this work.

References

1. T.F. Banchoff, *Beyond the Third Dimension: Geometry, Computer Graphics, and Higher Dimensions*, Scientific American Library, New York, 1990.
2. S. Feiner and C. Beshers, "Visualizing *n*-Dimensional Virtual Worlds with *n*-Vision," *Computer Graphics*, Vol. 24, No. 2, Mar. 1990, pp. 37-38.
3. C. Bajaj, "Rational Hypersurface Display," *Computer Graphics*, Vol. 24, No. 2, Mar. 1990, pp. 117-128.
4. F. Apéry, *Models of the Real Projective Plane*, F. Vieweg und Sohn, Braunschweig/Wiesbaden, Germany, 1987.
5. T.F. Banchoff, "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, New York, 1986, pp. 187-202.
6. G.K. Francis, *A Topological Picturebook*, Springer-Verlag, New York, 1987.
7. J.R. Weeks, *The Shape of Space*, Marcel Dekker, New York, 1985.
8. D.W. Brisson, ed., *Hypergraphics: Visualizing Complex Relationships in Art, Science and Technology*, AAAS Selected Symp., Vol. 24, Westview Press, Boulder, Colo., 1978.
9. K.V. Steiner and R.P. Burton, "Hidden Volumes: The 4th Dimension," *Computer Graphics World*, Feb. 1987, pp. 71-74.
10. S.A. Carey, R.P. Burton, and D.M. Campbell, "Shades of a Higher Dimension," *Computer Graphics World*, Oct. 1987, pp. 93-94.
11. A.J. Hanson and P.A. Heng, "Visualizing the Fourth Dimension Using Geometry and Light," *Proc. Visualization 91*, IEEE CS Press, Los Alamitos, Calif., 1991, pp. 321-328.
12. J.D. Foley et al., *Computer Graphics: Principles and Practice*, 2nd ed., Addison-Wesley, Reading, Mass., 1990.
13. A.H. Barr, "Superquadrics and Angle-Preserving Transformations," *IEEE CG&A*, Vol. 1, No. 1, Jan. 1981, pp. 11-23.
14. D. Hilbert and S. Cohn-Vossen, *Geometry and the Imagination*, Chelsea Publishing Co., New York, 1952.
15. B.K.P. Horn, *Robot Vision*, MIT Press, Cambridge, Mass., 1986.



Andrew J. Hanson is an associate professor of computer science at Indiana University. Previously, he worked in theoretical physics at a number of institutions and with the Perception research group in the SRI Artificial Intelligence Center. His research interests include the exploitation of computer graphics for visualizing abstract concepts in mathematics and physics, design of user interfaces for visualization applications, modeling methods in computer graphics and machine vision, object recognition, and artificial intelligence.

Hanson received his BA in chemistry and physics from Harvard College in 1966 and his PhD in theoretical physics from MIT in 1971. He is a member of AAAI, American Physical Society, ACM Siggraph, IEEE Computer Society, and Sigma Xi.

Pheng A. Heng is a PhD candidate in the Computer Science Department at Indiana University. His dissertation topic is interactive visualization tools for topological exploration, and his research interests include interactive mathematical visualization, scientific visualization, object-oriented graphics, and user interface design.

Heng received his BS in computer science in 1985 from the National University of Singapore, and his MS in computer science in 1987 and his MA in applied mathematics in 1988 from Indiana University. He is a member of ACM Siggraph and the IEEE Computer Society.

Address correspondence to Hanson at the Dept. of Computer Science, Indiana University, Bloomington, IN 47405, e-mail hanson@iuvax.cs.indiana.edu.