Interactive Visualization Methods for Four Dimensions

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

Abstract

Making accurate computer graphics representations of surfaces and volumes (2-manifolds and 3-manifolds) embedded in four-dimensional space typically involves complex and time-consuming computations. In order to make simulated worlds that help develop human intuition about the fourth dimension, we need techniques that permit real-time, interactive manipulation of the most sophisticated depictions available. We propose the following new methods that bring us significantly closer to this goal: an approach to high-speed 4D illuminated surface rendering incorporating 4D shading and occlusion coding; a procedure for rapidly generating 2D screen images of tessellated 3-manifolds illuminated by 4D light. These methods are orders of magnitude faster than previous approaches, enabling the real-time manipulation of high-resolution 4D images on commercial graphics hardware.

1 Introduction

The visualization of objects in four-dimensional space is an intriguing intellectual and computational problem. On the one hand, computer graphics is indispensable for making images that correspond accurately to 4D — since no 4D "sandbox" exists for us to play in, computer simulations are essential tools for experiencing four dimensions. On the other hand, intuitively appealing 4D rendering methods [2, 9, 7, 3, 4] typically have a high computational expense (up to hours per frame), which severely limits the degree to which such a system can support the development of human intuition by interactive exploration.

There are many potentially interactive methods for the subproblem of representing *surfaces* projected from 4D to 3D [1, 6, 3]. These include animated slices, rendering the 3D projection directly, exploiting pseudocolor or texture to indicate 4D depth, displaying 4D shadows, and using auxiliary vectors to compensate for the deficiency of surface tangent vectors. We have experimented with many of these methods, but in our opinion they lack the potential richness of the full 4D illuminated rendering approach that uses thickened 3-manifolds in place of the bare surfaces [3, 4].

Depictions of 3-manifolds have also been accomplished using a variety of techniques, such as projections and animations of hyperplane slices and strips[1]; again, we feel that full 4D illuminated rendering contains more potential information.

In this paper, we begin by addressing and solving the problem of choosing mathematical approximations that, combined with state-of-the-art graphics hardware, can be used to create a *completely interactive* system for visualizing surfaces and volumes embedded in four dimensions. To create our surface images, we generalize the 3D "teddy bear hair" algorithm [8] to four dimensions, obtaining theoretically justifiable approximations to the smooth and specular shading of a 2-manifold in 4D that has been thickened by the addition of a small shiny circle at each point of the normal space. We produce the effect of a full 4D depth-buffered rendering by providing an additional texture field marking the location of 4D occlusions. Finally, we introduce a very fast technique for displaying three-manifolds with 4D illumination. Our new approach shortcuts the original method of [3, 4], which creates an expensive volume image and then volume renders that to 2D. By combining a generalized ray-tracing technique with a hybrid scan-conversion method, we transform the volume rendering to an equivalent 3D rendering problem supported by graphics hardware.

All of these methods have been implemented to run on a Silicon Graphics Reality Engine, exploiting the high-performance scan-conversion and texture mapping capabilities whenever possible. We are thus able to interactively explore large classes of 2- and 3-manifolds embedded in 4D.

1070-2385/93 $3.00

2 Thickened Surfaces

We turn first to the question of rapidly producing qualitatively correct images corresponding to the thickened-surface volume images of Hanson and Heng [3, 4]. First, let us review the background and motivation for the original technique:

- The "film dimension" varies with the dimension of the space being viewed. Since projection along the camera ray creates an image of one dimension lower than the space, images of 2D space are lines, images of 3D space are planes, and images of 4D space are volumes with lit voxels, resembling a CT scan.
- Imaging using traditional computer graphics models requires that the objects being rendered have unique normal vectors. Thus, surfaces in 3D (typically boundaries of 3-volumes) can be rendered while mathematical curves cannot. In 4D, 3-manifolds (typically boundaries of 4-volumes) can be uniquely rendered, but surfaces and lines cannot. (For detailed descriptions of 4D diffuse and specular shading algorithms, see, e.g., [3, 4].)
- Informative images of curves in 3D can be produced by adding a circle to each point along the curve, thus generating a thickened curve, a cylindrical surface with a curve at the core. Corresponding images of surfaces in 4D can be created by adding a circle to each point of the surface's normal plane, giving a thickened surface that is actually a 3-manifold.

Complicated surfaces embedded in four dimensions may be rendered by thickening them, applying the analog of standard 3D illumination models (employing the now well-defined normal vectors of the resulting 3-manifold), and scan-converting the projection into a volume image. Just as z-buffering is used in 3D to handle occlusions, "w-buffering" can be employed in the 4D rendering process to handle 4D occlusions that occur when more than one scene point projects to the same point in the image volume. This volume image is then volume-rendered into a standard 2D computer screen image, preferably in stereo or as a rotating animation so that the viewer can perceive the internal structure of the volume.

Examples of a knotted sphere and an apparently knotted sphere rendered using this method for a video animation are shown in Figures 4a and 4b. These images are low resolution representations, but higher resolution was impractical: each image required up to half an hour of computation on a high-performance workstation, and took up to 48 megapixels of intermediate storage during the generation of the volume image, 16 megapixels each for the diffuse color, the specular color, and the w-buffer. The computational expense for even a one-minute animation (1800 frames) thus becomes staggering.

We are therefore obviously motivated to seek less expensive ways of producing such images. Returning to three dimensions for intuition, we note that the overall appearance of a rendered shiny wire is more or less independent of its thickness: it is precisely this observation that was exploited by Kajiya and Kay [8] to reduce the rendering of textures like hair to a computationally tractable problem while preserving essential qualitative features. We therefore attempt the analogous process for thickened surfaces in 4D. In the following, we show how to generalize the method of Kajiya and Kay to two-manifolds, so that we can effectively shrink the size of the thickening circle to a point without losing any essential image properties. The rendering problem is then partially reduced to a texture-mapping problem, which is ideally suited for real-time graphics hardware such as the Silicon Graphics Reality Engine.

2.1 Simplified Four-Dimensional Diffuse Reflection

In the model of diffuse reflection for 3D translucent hair chosen by Kajiya and Kay, the intensity depends only on the component of light lying in the normal plane, regardless of the viewpoint. If we imagine splitting the light vector into a component in the tangent space (a line) of the hair and a component in the normal space (a plane), their model keeps only the normal space component. The diffuse reflection may thus be expressed as the sine of the angle between the original light vector and the tangent to the hair.

This approach is easily generalized to surfaces in 4D space by realizing that the normalized 4D light vector $\hat{\mathbf{L}}$ separates naturally into two orthogonal components: one, $\overrightarrow{\mathbf{L}}_T$, in the tangent plane of the surface, and the other, $\overrightarrow{\mathbf{L}}_N$ in the normal plane. Again, a good approximation to the diffuse reflection of a surface with a small translucent circle attached is given by keeping only the normal space component $\|\overrightarrow{\mathbf{L}}_N\|$ of the light vector at each point. In practice, it is often easiest to compute this as the complement of the magnitude of the tangent-plane components of the light vector:

$$\|\overrightarrow{\mathbf{L}}_N\|^2 = 1 - \|\overrightarrow{\mathbf{L}}_T\|^2 . \qquad (1)$$

Thus, we adopt the following heuristic for the geometric component of diffuse surface shading in 4D:

$$D(\hat{\mathbf{L}}) = \|\vec{\mathbf{L}}_N\|. \tag{2}$$

As with the model of Kajiya and Kay, if we neglect shadowing, the intensity is independent of the viewpoint; this feature appears to be acceptable in our application. The complete diffuse lighting component Ψ_{diffuse} is described in terms of k_d, the color of the diffuse light, and $D(\hat{\mathbf{L}})$, the geometric term:

$$\Psi_{\text{diffuse}} = k_d D(\hat{\mathbf{L}}). \tag{3}$$

2.2 Simplified Four-Dimensional Specular Reflection

In [8], it is pointed out that the specular reflection from a thin cylinder is very nearly a cone. This cone may be determined by reflecting a single ray of light from a plane tangent to the hair at a point and then rotating that vector about the hair's axis. A lighting model is then adopted that determines the intensity in terms of the scalar product between the viewing vector and the closest vector lying on this cone. This is equivalent to replacing the dot product of the normal components of the light and view vectors by the product of the magnitudes of their normal components.

Surfaces in four dimensions permit a treatment of specular reflection that is closely parallel to the treatment of very thin hair in three dimensions. The tangent to a 4D surface is a plane instead of a line, but the normal space remains a plane. Thus we initially consider a specular coefficient that is decomposed into tangential and normal parts of the scalar product between the view vector $\hat{\mathbf{V}}$ and the reflected illumination vector $\hat{\mathbf{R}}$:

$$\hat{\mathbf{R}} \cdot \hat{\mathbf{V}} = \vec{\mathbf{R}}_T \cdot \vec{\mathbf{V}}_T + \vec{\mathbf{R}}_N \cdot \vec{\mathbf{V}}_N . \tag{4}$$

If we attach an infinitesimal circle lying in the normal plane to each point of the surface, points on this circle lie on small 3-manifold patches, which have unique normal vectors. Given this normal vector, we can find the reflection vector $\hat{\mathbf{R}}$. As with thickened 3D hair, we now simply replace the dot product of the normal components of the reflected light vector and the view vector by the product of the *magnitudes* of their normal components. This heuristic thus replaces the specular coefficient $\hat{\mathbf{R}} \cdot \hat{\mathbf{V}}$ by

$$S(\hat{\mathbf{R}}, \hat{\mathbf{V}}) = \vec{\mathbf{R}}_T \cdot \vec{\mathbf{V}}_T + \|\vec{\mathbf{R}}_N\| \, \|\vec{\mathbf{V}}_N\| . \tag{5}$$

This equation can be rephrased in terms of the light vector $\hat{\mathbf{L}}$, replacing $\vec{\mathbf{R}}_T$ with $-\vec{\mathbf{L}}_T$ and $\vec{\mathbf{R}}_N$ with $\vec{\mathbf{L}}_N$, yielding

$$S(\hat{\mathbf{R}}, \hat{\mathbf{V}}) = -\vec{\mathbf{L}}_T \cdot \vec{\mathbf{V}}_T + \|\vec{\mathbf{L}}_N\| \, \|\vec{\mathbf{V}}_N\| . \tag{6}$$

In Figure 1, we see how the vectors in the normal plane have been effectively replaced with their magnitudes, allowing us to completely describe the four-dimensional system with a three-dimensional figure.

If we now take k_s to be the color of the reflected light and Eq. (6) for $S(\hat{\mathbf{R}}, \hat{\mathbf{V}})$ to be the heuristic Phong-like specular term (with a floor of zero) to be raised to a power p, we find the intensity

$$\Psi_{\text{specular}} = k_s S(\hat{\mathbf{R}}, \hat{\mathbf{V}})^p . \tag{7}$$

2.3 4D Occlusions

For 4D surfaces such as knotted spheres, whose 3D projections contain massive self-intersections, additional features are needed in the texture-map generation algorithm in order to retain all the qualitative properties of Figure 4. Each 3D self-intersection corresponds to a place where one surface patch occludes another along the 4D line of sight, so correct w-buffer emulation should permit only the nearest patch to be seen. When such occlusion marking is desired, one may check for parts of the surface in 4D that have 3D coordinates very near those of other parts in a particular 4D⇒3D camera projection. Then the distances from the conflicting areas to the camera focal point are compared, and the more distant area is painted with transparent or opaque black; the areas nearer the camera can also be painted in a distinctive color for emphasis. This method gives us the precise analog of a traditional knot-crossing diagram such as the one shown in Figure 2a, where a substantial section of each occluded line segment is removed from the drawing. Figure 2b shows our corresponding picture for the spun trefoil, a simple knotted sphere, projected to 3D from a particular 4D viewpoint, cut away to reveal the interior.

The following features of 4D occlusion algorithms should be noted:

- Certain classes of objects, such as spun knots in 4D, permit simple calculation of the occlusion texture due to the separability of occlusion points into a small set of 2D intersection problems. These properties can be conditionally dependent upon the chosen 4D viewpoint.

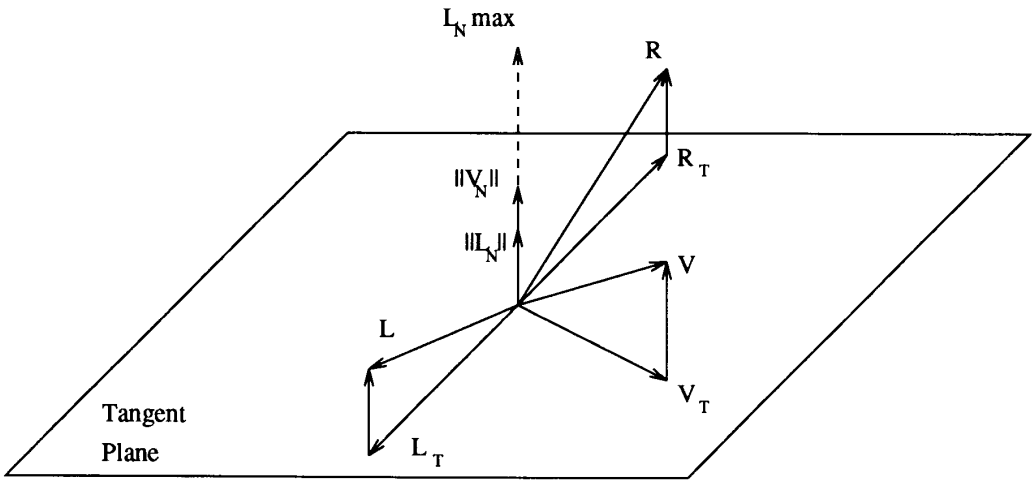

Figure 1: The tangent plane of a surface at a particular point showing the four dimensional vectors and their relevant tangent and normal components. $\overrightarrow{L}_N$ max represents the value the light's normal component would assume in this diagram if $\overrightarrow{L}_T$ vanished.

- For high-quality transparency, the texture polygons must be rendered in back-to-front order with the help of a BSP tree, a time-consuming preprocessing requirement.

- For general 4D objects and general 4D viewpoints, the occlusion texture must be computed by recalculating the entire self-intersection map in the new 3D projection. A variety of methods can be used, ranging from brute force to BSP trees to an augmented z-buffer that keeps a database of 4D depths as each polygon is scan-converted from 3D to 2D.

- Some 4D-viewpoint-independent features assisting in the occlusion calculations can be maintained in a 4D BSP-tree data structure. In particular, such a data structure can be used in a viewpoint independent hidden-volume calculation during the projection to 3D, much as the 3D BSP tree is used for viewpoint-independent hidden surface elimination.

- 4D self-shadowing is potentially important when complex surfaces are illuminated by 4D light. Such shadows can be computed by creating a thick occlusion-marking texture derived by using the point light source as the 4D focal point and projecting to 3D.

2.4 Results

In Figures 5a,b, we show the results of combining the approximations for the diffuse, specular, and occlusion-tagged texture that we have developed as interactive alternatives to the fully-correct volume renderings shown in Figures 4a,b for the spun trefoil knotted sphere and the apparently (but not) knotted twist-spun trefoil. These images have the following remarkable properties:

- Once a 4D viewpoint is chosen and occlusion preprocessing (if required) is performed, these images can be rotated arbitrarily in 3D, and the 4D lighting vector can be changed arbitrarily; redisplay time of a 50,000 polygon tessellation on the SGI Reality Engine occurs is less than one second, including completely recomputing the shading portion of the texture map and transferring it to texture memory. This is a speed increase of up to *three orders of magnitude* over the original method.

- The resolution of the image can be as detailed as that of a full 1000×1000 workstation screen with no significant performance penalty. The contrast in precision is plainly evident, and much additional detail can be seen.

- The important qualitative features of Figure 4 are completely preserved in Figure 5, including the

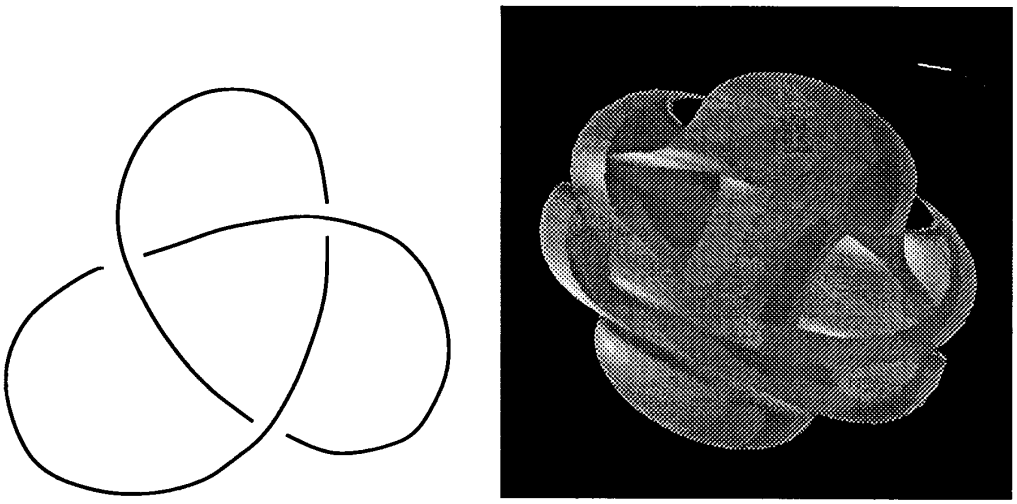

Figure 2: (a) Typical representation of a 3D knot projected to a 2D image that uses symbolic cutaways to represent occlusions and reveal the full structure of the knot. (b) A slice into the interior of the analogous representation of the spun-trefoil knotted sphere; three entire circular ribbons are removed from the 3D surface by painting them a transparent black texture. These sections are farther away from the 4D camera than the normally shaded parts of the surface with which they intersect in the 3D projection.

clear depiction of knot-crossing regions in four-dimensional space.

3 Three-Manifolds

There are many interesting 4D objects that are in fact 3-manifolds, not surfaces, and therefore cannot be rendered using the acceleration methods described so far. In particular, we have previously generated images of objects such as 3-spheres, 4D superquadrics, hypercubes (tesseracts), and 3D scalar fields (equivalent to 4D elevation maps) using the full volume-image/volume-rendering methods [4, 5]. What alternatives are available for interactive manipulation of objects such as these?

3.1 Plane-Tracing

We begin by describing a fundamental technique that takes us the first step towards simplifying the computation of the twice-projected 2D image of any 3-manifold in 4D that has been decomposed into tetrahedral volumes (precisely analogous to decomposing a surface in 3D into triangles). To understand the concepts, first consider a 3D world projected to a 2D image and imagine that our desired result is a 1D image given by the *sum* of the pixels in each column of the 2D image array. In Figure 3a, we show that the contribution of a polygon to the image intensity in each 1D pixel can be computed from the *length* of the polygon line segment intersecting the ray plane of the column, projected to the image plane. The contribution of the polygon to the 1D pixel is the integral of the intensity over the line; if the intensity is constant (e.g., planar faces and distant light), the contribution is the product of the intensity and the length of the line. This procedure might be called "plane-tracing" — ray-tracing with a *planar ray*; it produces a 1D image directly and eliminates 2D rendering from the process of imaging 3D objects tessellated into planar faces. Adding perspective, smoothly interpolated shading, and occlusion is in principle straightforward.

Four Dimensions. In 4D, oriented 3-manifolds form boundaries of objects (polytopes) whose interiors are 4-volumes; it is these 3-manifolds that are projected to the image volume and rendered. Since we can see only one particular 2D view of the volume image at a time, we can exploit a shortcut analogous to the one just described in 3D: determine a 2D pixel value directly from the projected lines found by intersecting a planar ray through a column of the volume image with each tetrahedral volume. The contribution of a tetrahedron (hyperface) to such a pixel is the integral

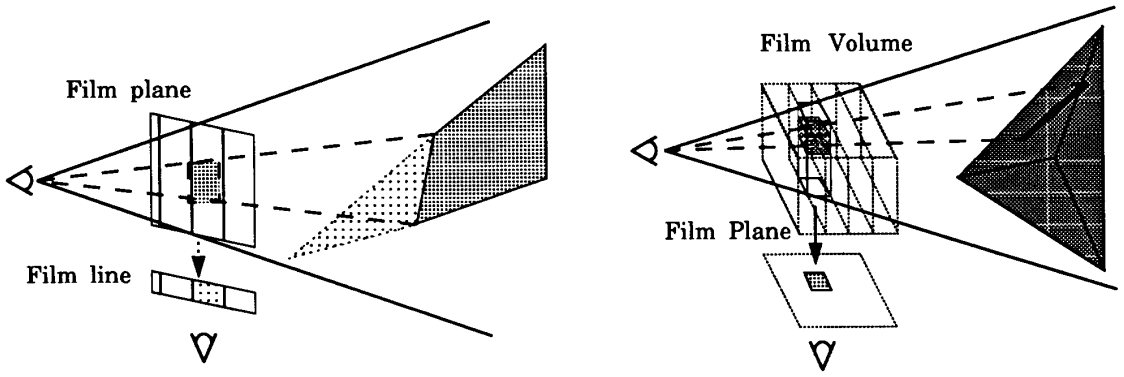

Figure 3: (a) Schematic diagram of the construction of a column-summed image of a 3D object without an intermediate 2D image; only the projected length of the line where a triangle intersects the planar ray need be computed. (b) Schematic diagram showing how a solid tetrahedron in 4D contributes an intensity to its 2D image corresponding to the length of a single line.

of its intensity over the projected length of the intersection line in the ray plane; if the hyperface intensity is constant, one need only multiply by the projected line length. A schematic diagram of this process is given in Figure 3b.

Speed-up Approximation. We can achieve a remarkable acceleration in 4D volume rendering whenever the intensity changes across portions of the projected tetrahedra are effectively linear, e.g., when the hyperface intensity is constant. Then we may reformulate the problem as an equivalent 3D Gouraud shading task. We need only plane-trace the locations of each tetrahedron's vertices and the projected crossing points of two opposite edges (when such a crossing occurs), assign intensities as described above to these 2D points, and let standard bilinear interpolation algorithms (e.g., in hardware) fill in the rest of the 2D image intensities. Using an additive alpha-blend mode with no z-buffering produces excellent representations of complex tessellated 3-manifolds in a fraction of a second. Similar accelerations can be achieved for non-constant hyperface intensity provided the integral of the intensity along the projected line can be simply computed and interpolated among 2D projected vertices.

3.2 Geometry

We note that the geometry involved in this 4D construction is highly counterintuitive. We are used to thinking in 3D that planes intersect in a common line and that a solid intersects a plane to form an area patch in the plane. In 4D, planes intersect in *one point*: a plane is described in 4D by 2 constraint equations, so 2 planes give four constraints, and their mutual solution is a point. Furthermore, a plane intersects a hyperplane (volume) in a *line*: two equations for the plane plus one equation for the hyperplane give three equations in 4D, leaving only a line, not a surface, in the intersection.

Let us define the "planar ray" contributing to pixel (i,j) as a plane bounded by the lines from the camera's focal point $\vec{\mathbf{C}}$ to the bottom point $\vec{\mathbf{S}}_0$ and to the top point $\vec{\mathbf{S}}_1$ of the corresponding column of the volume image. Any point in that plane has barycentric coordinates (α,β), e.g.,

$$\vec{\mathbf{x}}_{\alpha,\beta} = \vec{\mathbf{C}} + \alpha(\vec{\mathbf{S}}_0 - \vec{\mathbf{C}}) + \beta(\vec{\mathbf{S}}_1 - \vec{\mathbf{C}}). \quad (8)$$

If $\{\vec{\mathbf{x}}_0, \vec{\mathbf{x}}_1, \vec{\mathbf{x}}_2, \vec{\mathbf{x}}_3\}$ are the four 4D points of one of a set of tetrahedral "faces" bounding a 4D volume, we may examine each of the four associated planar triangles in turn to see if they intersect the chosen viewing wedge at interior points or whether they miss (like a triangular face in 3D missing a scan line or an image column in Figure 3a).

For example, we can write the parametric form of one of the four faces of a tetrahedron as

$$\vec{\mathbf{X}}_{\mu,\nu} = \vec{\mathbf{x}}_1 + \mu(\vec{\mathbf{x}}_2 - \vec{\mathbf{x}}_1) + \nu(\vec{\mathbf{x}}_3 - \vec{\mathbf{x}}_1). \quad (9)$$

If we then set $\vec{\mathbf{x}}_{\alpha,\beta} = \vec{\mathbf{X}}_{\mu,\nu}$, Eqs. (8) and (9) form a set of four equations in four unknowns which we solve to find their common point. If $0 \leq \alpha \leq 1$,

$0 \leq \beta \leq 1$, the point is potentially visible to the camera. If $0 \leq \mu \leq 1$, $0 \leq \nu \leq 1$, $0 \leq 1 - \mu - \nu \leq 1$, then the point lies within the face of the tetrahedron. Either two or zero such mutual solution points will exist among the four tetrahedral faces. If two intersections are found, we draw the line joining them, project the line to the volume image and add a contribution to the 2D image intensity equal to the integral of the projected voxel values over the projected length. For the special case of constant voxel intensities within the entire tetrahedron, we just have

$$\Delta I = I_0 \cdot \{\text{projected line length}\}. \qquad (10)$$

3.3 Results

In Figure 6, we show the results of applying this approach to the rendering of a cross-eyed stereo pair of a tesseract. Our current implementation allows the user to rotate the 4D light, 3D object orientation, or 4D object orientation at more than 30Hz for a 1000×1000 image of a tesseract (up to 20 visible tetrahedra in a given view) on an SGI Reality Engine system. The update time is expected to be essentially linear in the complexity of the tessellation. This overall approach is valid for any object well-approximated by a convex polytope, including simple 3D scalar fields [5]. Adding approximate smoothly shaded interpolations is in principle straightforward (e.g., by transforming to an equivalent Phong interpolation instead of a Gouraud interpolation). Nonconvex or multiple objects would need additional attention to avoid the display of scene portions that should be occluded; 4D shadows should in principle be incorporated as well in such scenes. The approach to rendering 3-manifolds described here is easily formulated as a parallel problem; thus we expect to be able to handle increasingly complex interactive 3-manifold rendering in the next generation of mathematical visualization systems.

Acknowledgments

We thank Pheng Heng, Hui Ma, and Lie-Hwang Hwang for their contributions to this investigation. We also thank David Banks for suggesting to us the relevance of reference [8]. We gratefully acknowledge the facilities support of Indiana University and of CICA (the IU Center for Innovative Computer Applications). This research was supported in part by NSF grant IRI-91-06389.

References

[1] T.F. Banchoff, *Beyond the Third Dimension: Geometry, Computer Graphics, and Higher Dimensions*, (Scientific American Library, New York, 1990).

[2] S.A. Carey, R.P. Burton, and D.M. Campbell, "Shades of a Higher Dimension," *Computer Graphics World*, pp. 93–94, October 1987.

[3] A.J. Hanson and P.A. Heng, "Visualizing the Fourth Dimension Using Geometry and Light," in the Proceedings of *Visualization '91*, San Diego, CA, Oct 22–25, pp. 321–328, 1991.

[4] A.J. Hanson and P.A. Heng, "Illuminating the Fourth Dimension," Computer Graphics and Applications, **12**, No. 4, pp. 54–62 (July, 1992).

[5] A.J. Hanson and P.A. Heng, "Four-Dimensional Views of 3D Scalar Fields," in Proceedings of *Visualization '92*, Boston, MA, Oct 21–23, 1992, pp. 84–91 (IEEE Computer Society Press, Los Alamitos, CA, 1992).

[6] C. Hoffmann and J. Zhou, "Some Techniques for Visualizing Surfaces in Four-Dimensional Space," Computer-Aided Design, **23**, pp. 83–91 (1991).

[7] S. Hollasch, "Four-space visualization of 4D objects," Master's Thesis, Arizona State University, August 1991.

[8] J.T. Kajiya and T.L. Kay, "Rendering Fur with Three Dimensional Textures," in the Proceedings of Siggraph '89, Computer Graphics **23**, pp. 271–280 (1989).

[9] K.V. Steiner and R.P. Burton, "Hidden Volumes: The 4th Dimension," *Computer Graphics World*, pp. 71–74, February 1987.

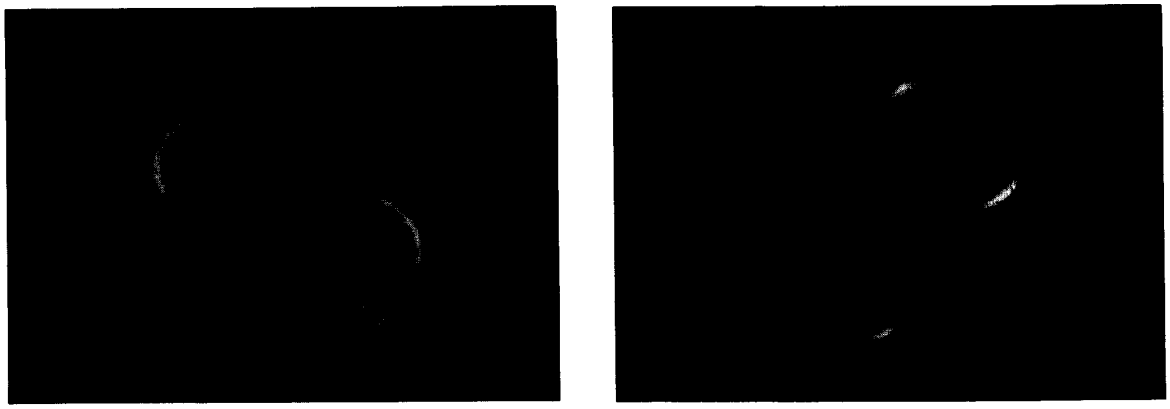

Figure 4: Volume rendering of volume images produced using full 4D scan-conversion of thickened surfaces (Hanson and Heng [4]). (a) Spun trefoil. (b) Twist-spun trefoil.

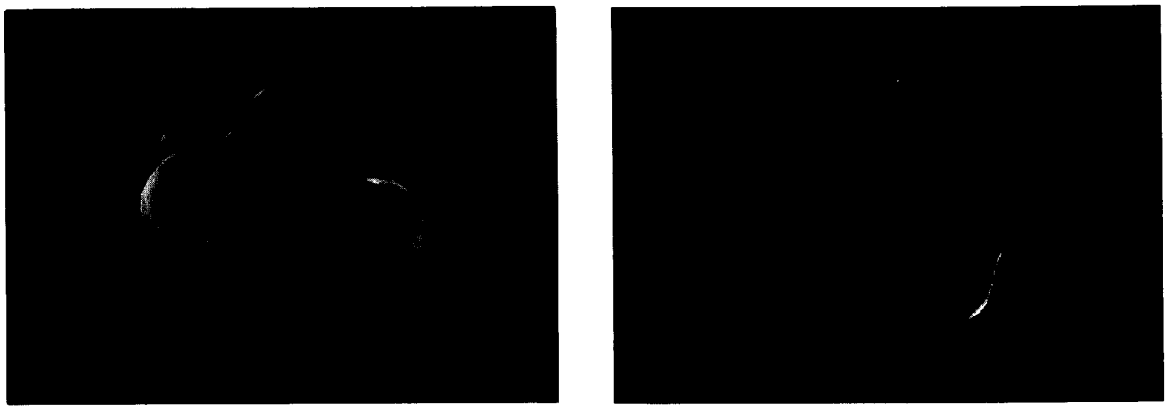

Figure 5: Images of surfaces created interactively using the "4D thin hair" method introduced in Section 2. (a) Spun trefoil. (b) Twist-spun trefoil.

Figure 6: Cross-eyed stereo pair of a hypercube in 4D created interactively using the method of Section 3. The front, bottom, and right sides of the blue cube face the viewer; the magenta cube is a "roof" viewed from below.

(See color plates, p. CP-19.)

Interactive Visualization Methods for Four Dimensions, A.J. Hanson and R.A. Cross, pp. 196-203.

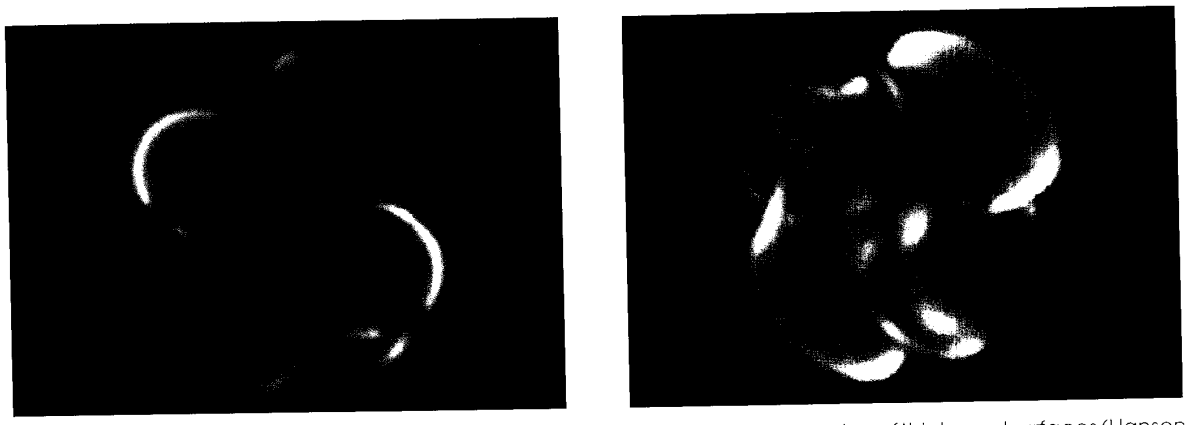

Figure 4: Volume rendering of volume images produced using full 4D scan-conversion of thickened surfaces (Hanson and Heng (4)). (a) Spun trefoil. (b) Twist-spun trefoil.

Figure 5: Images of surfaces created interactively using the "4D thin hair" method introduced in Section 2. (a) Spun trefoil. (b) Twist-spun trefoil.

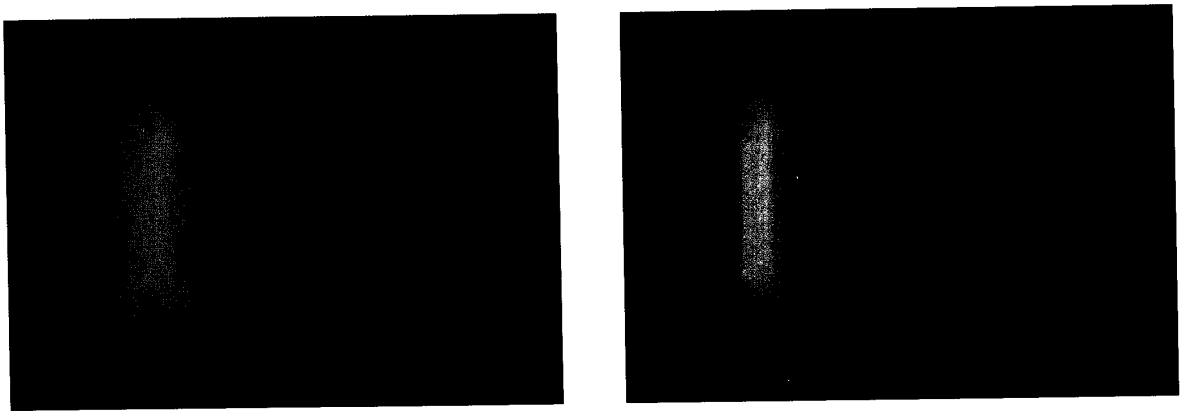

Figure 6: Cross-eyed stereo pair of a hypercube in 4D created interactively using the method of Section 3. The front, bottom, and right sides of the blue cube face the viewer; the magenta cube is a "roof" viewed from below.