Pacific Graphics (2013)
B. Levy, X. Tong, and K. Yin (Editors)



# Interactive Exploration of 4D Geometry with Volumetric Halos

W.M. Wang[1], X.Q. Yan[2], C.W. Fu[2], A.J. Hanson[3] and P.A. Heng[1,4]

[1]The Chinese University of Hong Kong
[2]Nanyang Technological University
[3]Indiana University
[4]Shenzhen Institutes of Advanced Technology

**Abstract**

*Halos have been employed as a compelling illustrative hint in many applications to promote depth perception and to emphasize occlusion effects among projected objects. We generalize the application of halo methods from the widely-used domain of 2D projections of 3D objects to the domain of 3D projections of 4D objects. Since 4D imaging involves a projection from 4D geometry (such as a surface with 4D vertices) to a 3D image, such projection typically produces intersecting surfaces, and thus occlusion phenomena result in apparent curves in 3D space. Adding volumetric halos to the surfaces then gives useful information about the spatial relations of intersecting surfaces, and allows a more accurate perception of the geometry. A typical application is knotted spheres embedded in 4D, and the volumetric halos perform the same function as traditional knot diagrams do in 2D drawings of 3D knotted curves. In addition, we design a series of GPU-based algorithms to achieve real-time updating of the halo-enhanced image when the geometry is interactively rotated in 4D.*

Categories and Subject Descriptors (according to ACM CCS): [I.3.8]: Computer Graphics—Applications; [I.3.6]: Computer Graphics—Interaction techniques

## 1. Introduction

Occlusion is an important depth cue, providing information about relative depth, proximity, and spatial arrangements of the objects in a given view. A refinement of occlusion is the *crossing diagram* of a set of curves embedded in 3D space, an illustrative technique that adds schematic emphasis to the 2D image by cutting away small neighborhoods of each occluded segment; viewer perception of the 3D features in the 2D image is often significantly enhanced by this method. In this paper, we explore the generalization of 3D crossing diagrams to produce schematically emphasized occlusion diagrams for surfaces in 4D space projected to a 3D image. There are many methods to produce occlusion diagrams of 3D curves as well as 4D surfaces. We confine our attention here to the *halo methods* [ARS79]. Just as ordinary knots are a common useful domain for crossing diagrams in 2D images of 3D objects, knotted spheres [HC93, CH94, CFHH09, Ino13] are a classic example of a 4D visualization problem that profits from the exploitation of crossing diagrams.

Illustrative rendering techniques such as halos and image sharpening can significantly improve the perception of depth and occlusion. Typical halo methods [ARS79, IG98] render halos around lines or object silhouettes, emphasizing

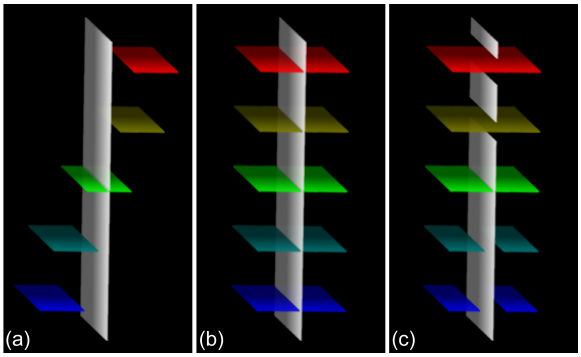


**Figure 1:** *Surface patches with different 4D depth values in w projected to a 3D image: (a) oblique 4D view projected along direction* $(x,y,z,w) = (1,0,0,1)$*, which shows the physical separation in w; (b) 4D view along* $(0,0,0,1)$*, producing 3D intersections, without halo; (c) 4D view as (b) enhanced by our depth-dependent volumetric halos.*





DOI: 10.2312/PE.PG.PG2013short.001-006

the occlusions among projected objects on the image screen. Similarly, we can employ any available depth information to sharpen and emphasize the properties of an object in its rendered image [LCD06], and to locally enhance the image color and contrast around object silhouettes. Thus we can both emphasize foreground objects to make them more recognizable, and reveal the spatial relations among projected objects. These rendering techniques are also commonly used by artists to strengthen object perception in their drawings, especially for illustrative figures in a technical context.

Following the spirit of [ARS79, BG07, EBRI09], we design halo-based methods to annotate the rendering of 4D geometry after projecting them from 4D to 3D. Figure 1 presents an elementary example of the desired results for five 4D colored surface patches whose 3D projections intersect the white patch, but that have different 4D depth values relative to the white patch. In addition to explaining how to generate and render the volumetric halos, we also show how this potentially slow and difficult rendering process can be made interactive by exploiting GPU-based methods, thus facilitating intuitive user exploration of the fourth dimension.

**Contributions of This Work.**

- We present a comprehensive volume-rendering-based approach to the creation of halo structures for surfaces embedded in 4D and projected to a 3D image.
- We extend the fast GPU-based method of Chu et al. [CFHH09] to include volumetric halos to support interactive exploration of 4D haloed occlusion and depth.
- We develop novel algorithms to resolve the problem of tetrahedron mismatch during the creation of auxiliary halo 3-manifolds needed to implement our approach.

## 2. Related Work

**Illustrative Visualization.** While non-photorealistic rendering techniques [GGSC98] depict data in an expressive way, illustrative visualization [DMNV12] focuses also on the use of visual emphasis to highlight relevant features of interest in the visualization. This is particularly useful when exploring massive or complicated data. Apart from halos, several illustrative visualization methods have been proposed. Diepstraten et al. [DWE03] explored different approaches to generate cutaway illustrations at interactive speed with a small set of rules. Li et al. [LRA*07] further considered a geometry-aware approach supporting interactive authoring in cutaway illustrations of complex 3D models. Karpenko et al. [KLMA10] recently illustrated shape and internal structures in complicated mathematical surfaces using exploded view diagrams by partitioning the surfaces into parallel slices. Meanwhile, Hummel et al. [HGH*10] examined various illustrative techniques on integral surfaces to convey both shape and directional information.

**Enhancing Visualization with Halos.** The utilization of halo methods in computer graphics (other than knot diagrams) goes back at least to the late 1970's when Appel et al. [ARS79] proposed using halos to emphasize the occlusions between lines and surfaces in various geometries. After that, Interrante and Grosch [IG98] introduced several novel halo-based techniques to illustrate 3D flow using volume line integral convolution to convey directional information. Schussman et al. [SMSE00] proposed various perceptually-effective techniques to visualize magnetic field data from the DIII-D Tokamak, while Ebert and Rheingans [ER00] combined the strengths of direct volume rendering with the expressive power of non-photorealistic rendering to create volume-based feature halos. Later, Luft et al. [LCD06] developed a halo-like approach to enhance the perceptual quality of images by using depth information, and Bruckner and Gröller [BG07] proposed a wide variety of GPU-based halo effectsto enhance the depiction of volumetric data. More recently, Everts et al. [EBRI09] suggested depth-dependent halos for effective illustrations of dense line data.

**Four-dimensional Visualization.** Using computer graphics and visualization methods, we can interactively explore geometry in four (or more) dimensions [HCV52], even though such geometry cannot be directly constructed or observed in our physical world. The early work of Noll [Nol67] in the late 1960's stimulated awareness of the issues and possible methods for representing high-dimensional objects using computer graphics. Basic graphics methods for representing 4D objects, initially as wire frames, were then developed for geometric objects of interest and converted into images and animated projections by numerous authors [Ban90, Hol91]. Moreover, Hoffmann and Zhou [HZ91] developed tools to visualize 2D surfaces embedded in 4D space. Hanson and Heng [HH92] studied the illumination effects for volumes as well as thickened curves and surfaces. Banks [Ban92] explored ways to visualize 4D depth information with methods like ribboning, transparency, lighting, and texturing.

## 3. Overview of Our Approach

Figure 2 outlines the volumetric halo process. We start with two surfaces that are separated in 4D space but appear to intersect each other after the 4D to 3D projection (Figure 2(a)). To determine the halo regions, we first apply a thickening process to the projected surfaces to extend them into a 3D volume (Figure 2(b)). Note that our thickening process is very different from that of Hanson and Heng [HH92], designed to support 4D lighting. Our thickening is done in 3D screen space rather than in 4D space, which is view-dependent, and more efficient for haloing, though it supports only a heuristic variant of rigorous 4D lighting. Hence, we need to perform the thickening in real-time for each change in viewpoint or projection. After the thickening, we can construct tetrahedra to populate the space occupied by the halos, and fill the depth buffer with 4D depth values during the rasterization process (Figure 2(c)). Finally, we can render the projected surfaces in the 3D screen space with a proper occlusion test against the depth buffer, thereby producing volumetric halos in the visualization (Figure 2(d)).

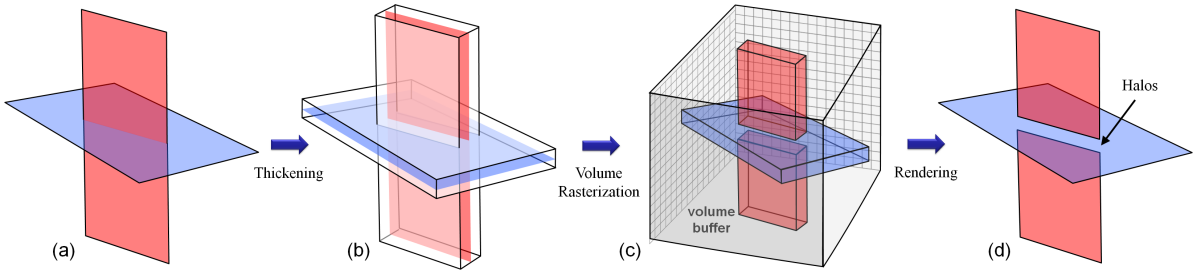


**Figure 2:** *Halo illustrations for surfaces in 4D: (a) projection from 4D space to 3D screen space resulting in intersecting surfaces; (b) view-oriented thickening; (c) rasterizing the thickened volumes by multi-slicing; (d) the final halo illustrations.*

## 4. The Illustrative Visualization Pipeline

### 4.1. 4D Geometry Generation

We design a *vertex program* to generate the 4D geometry directly on the GPU. Similar to the case of mathematical surfaces in 3D, we can employ *uv*-parametrized equations to define surfaces in 4D. In practice, we program these equations in the *vertex program*, thus allowing us to send only a small number of parameters to the GPU. Furthermore, instead of uniformly sampling the *uv* space, we adaptively sample and construct the underlying surface according to its local curvature and pre-compute two lists of $u$ and $v$ coordinates (with their connections to form triangles) so that the *vertex program* can help compute the corresponding 4D vertex positions. For each vertex on the surface, we compute a pair of 4D tangent vectors, say $\vec{t}_1$ and $\vec{t}_2$, and project them to the 3D screen space. Finally, all these per-vertex data attributes are stored using a vertex buffer object (VBO) to reduce the burden of transferring data between the CPU and the GPU.

### 4.2. Transformation and Projection

Just as 3D rotations produce different 2D screen images of a 3D scene, 4D rotations of a 4D scene produce different 3D volume images. Note that while 3D rotations have 3 degrees of freedom (e.g., rotations in the planes $yz$, $zx$, and $xy$), 4D rotations have 6 degrees of freedom (e.g, rotations in the planes $wx$, $wy$, $wz$, $yz$, $zx$, and $xy$). Moreover, 4D projection can be orthographic or perspective just as in 3D. These 4D transformations and projections are per-vertex operations (Figure 3(a)) implemented with a specific *vertex program* that computes the operations and stores the results along with 4D depth information.

### 4.3. Thickening Mechanism

Mathematically, we can derive the thickening direction $\vec{n}$ by taking a 3D cross product of the two projected surface tangents $\vec{t}_1$ and $\vec{t}_2$ after applying any transformations. There is no surviving 4D component at this point, resulting in:

$$\vec{n} = \vec{t}_{1xyz} \times \vec{t}_{2xyz} \,. \tag{1}$$

Note that the thickening direction $\vec{n}$ is dependent on the 4D viewing projection because the 3D projections of $\vec{t}_1$ and $\vec{t}_2$ change with the 4D rotation during interactive exploration.

During the thickening process, we create two vertex instances (positive and negative) for each vertex on the projected surface (Figure 3(b)). Hence, we can form two offset surfaces for each group of vertex instances. After that, we attach a per-vertex parameter $t$ that takes a value of 1 or 0 for vertex instances on the positive or negative offset surface, respectively. Now, we can also define an interactively-controllable parameter $\rho$ to adjust the size of the halos:

$$\vec{v}_{new} = \vec{v}_{orig} + (2t - 1)\,\rho\,\vec{n}\,, \tag{2}$$

where $\vec{v}_{orig}$ and $\vec{v}_{new}$ are the original and extruded coordinates on the projected and offset surfaces, respectively.

### 4.4. Modeling Halos with Tetrahedra

After the thickening, we next construct tetrahedra to model the volumetric space occupied by the halos. We join each pair of corresponding triangles on the two offset surfaces into a triangular prism, and then decompose the prism into three tetrahedra (Figure 3(c)). Altogether, there are eight possible ways of decomposing the faces of a triangular prism into triangles, only six of them leading to valid tetrahedra; applying a certain decomposition to all prisms without checking sharing faces will typically produce face mismatches similar to the T-join problem in triangular meshes (Figure 4), which will be discussed in subsection 5.1.

We define a dedicated *geometry program* in the GPU to perform the thickening and tetrahedralization. The *geometry program* takes a single triangle with three ordered vertices as input and ultimately outputs the geometry as a list of three tetrahedra for each input triangle. These tetrahedra are then stored in another *VBO* before the next step. Having another *VBO* as a temporary buffer can avoid unnecessarily re-generating the tetrahedra for different planar slices when rendering the halos, thus improving the overall performance.

### 4.5. Depth-dependent Volumetric Halos

Our volumetric halos adopt the depth-dependent property from Everts et al. [EBRI09] so that we can vary the gap size according to the difference of 4D depth values between the projected surfaces. To produce this depth-dependent effect with volumetric halos, we need to properly compute a 4D depth value for every voxel inside the halo volumes.

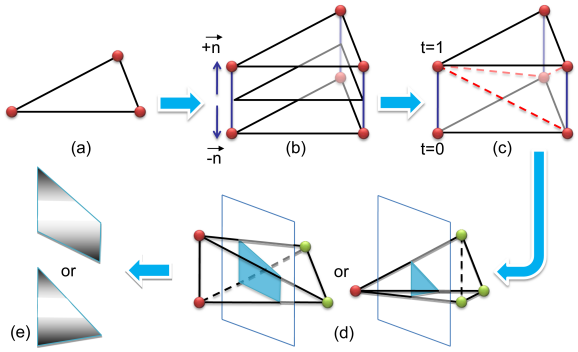


**Figure 3:** *Modeling halos from a triangle primitive: (a) the triangle is transformed and projected into 3D screen space; (b) thickening the triangle by offsetting it to both positive and negative sides of $\vec{n}$; (c) decomposing the extruded volumes into tetrahedra; (d) voxelizing the tetrahedra by planar slicing; (e) generating and rendering the halos (gray color).*

This is analogous to the case of depth-dependent halos in 3D graphics, where we increase the 3D depth values across the winged regions around a 1D line on the 2D screen. By using this depth increment strategy, we can fill the depth values for regions occupied by the halos, and use these extrapolated 4D depth values to eliminate any object fragment that is shadowed by the halos. We increase the 4D depth value of each voxel (within the halo volumes) according to its offset distance from the projected surface. Given a point $p$ with a 4D depth value of $d_{orig}$ on the projected surface, we use the following formula to compute the new 4D depth value $d_{new}$ at any point extruded along $\vec{n}$ from $p$:

$$d_{new} = d_{orig} + max(0, |2t - 1| - \delta t)\, \delta d\,, \qquad (3)$$

where $\delta d$ controls the drop-off rate of the depth value, and $\delta t$ controls the size of the middle region having the same constant 4D depth value as the point $p$.

### 4.6. Producing the Halo Visualization

We employ a multi-slice approach to generate and render the halos by composing the 2D slice images from back to front to produce the overall volume rendering. Figure 3(d) illustrates the slicing process, which is performed on each tetrahedron by another *geometry program*. Given a slicing plane and a tetrahedron, we first compute the points where the tetrahedron edges intersect the slicing plane. Since our slicing planes are all perpendicular to the $z$-axis in the 3D screen space, this step can be quickly done by computing the difference between the slicing plane and the $z$ components of tetrahedron vertices and checking the signs. After we identify the intersecting edges (with opposite signs for the two end points), we can interpolate various per-vertex parameters along the related edges, including the 4D depth value and the $t$ value. Then, we can output a triangle strip from the *geometry program* representing the intersection footprint, which could either be a quad or a triangle.

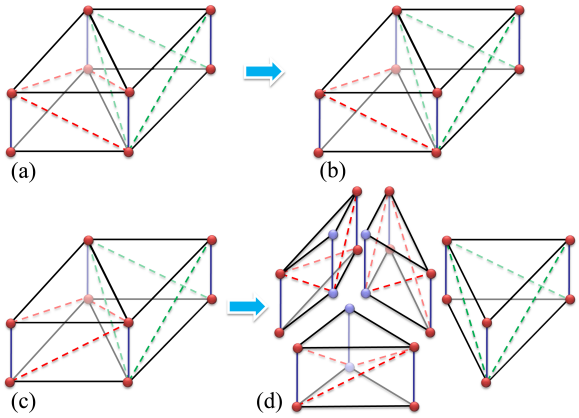


**Figure 4:** *Resolving face mismatches between neighboring triangular prisms: (a) a common type of face mismatch; (b) fixable by reversing the construction of the red line in the middle rectangle; (c) another type of face mismatch that is not fixable by (b); (d) fixable by adding new vertices and reconstructing the red prism with a new decomposition.*

After that, we interpolate the per-vertex attributes over the footprint triangle in the rasterization, so that each pixel (voxel) fragment receives its own piece of interpolated data that is to be further processed by a *fragment program*. By then, we can compute Equation (3) and fill the depth buffer properly with 4D depth values corresponding to the halo volumes on the slice being considered. By checking the interpolated $t$ value at the fragment (its proximity to 0.5), we can then properly render the projected surface with halos, and generate a slice image with anti-aliasing. At the end, we can progressively compose the slice images to produce the overall volume rendering.

## 5. Implementation and Results

### 5.1. Implementation Issues

As mentioned in subsection 4.4, if we randomly decompose all the triangular prisms, we will likely encounter face mismatches between some neighboring prisms; typically, there are two types of face mismatches (Figure 4(a) and 4(c)). The challenge here is that each triangular prism may contact as many as three other prisms constructed between the two offset surfaces. Hence, when we try to match the triangle pattern between a certain pair of prisms, we need to keep the matched triangle patterns for the other four faces. Otherwise, we may create new mismatches somewhere else when we fix the face mismatch between the current prism pair.

In the first case (Figure 4(a)), we can fix the face mismatch by simply changing the way we decompose the red prism (Figure 4(b)). The outer faces on the red prism are not changed but the triangle pattern on the sharing face can be flipped. In the second case (Figure 4(c)), it is more complicated because there is no way of decomposing the red prism that can allow us to resolve the face mismatch while keeping

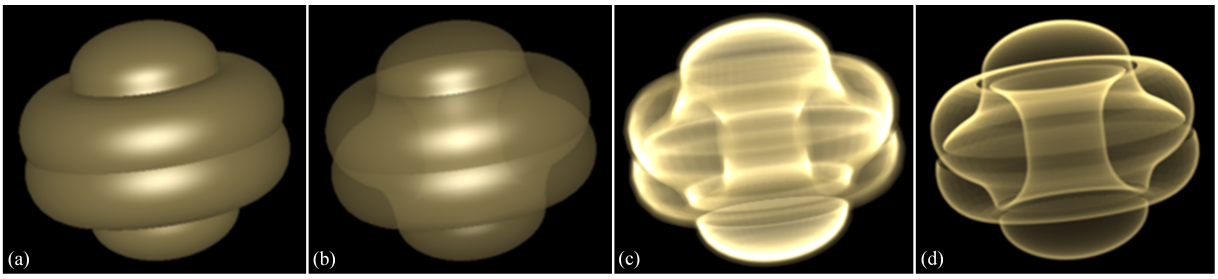


**Figure 5:** *Spun-trefoil knot: (a) traditional rendering; (b) translucent rendering; (c) result from [CFHH09]; (d) our result.*

**Table 1:** *Performance of our visualization pipeline (FPS).*

| Models / Num. Slices | 64 | 128 | 256 | 512 |
|---|---|---|---|---|
| Quadratic Fermat | 69.97 | 57.20 | 43.73 | 25.64 |
| Spun-trefoil knot | 69.16 | 55.31 | 42.67 | 24.30 |
| Twist-spun-trefoil knot | 65.45 | 49.35 | 35.32 | 20.98 |

the triangle patterns on the outer faces. To resolve this problem, we need to introduce new vertices to subdivide the red prism into three smaller prisms (Figure 4(d)) so that we can keep the desired triangle patterns on the outer faces that are originally on the red prism.

### 5.2. Performance

To evaluate the performance of our illustrative visualization pipeline, we tested it on a desktop PC with Intel(R) Core(TM) i7 CPU 960 3.20GHz and a Geforce GTX580 graphics card. Table 1 shows the performance results in frames-per-second (FPS). *Num. Slices* represents the number of slicing planes in the multi-slice processing, which is an important factor affecting both the overall performance and image quality; we usually need around 256 slicing planes to achieve reasonable quality for most results presented here. It is clear from the table that we can achieve real-time performance even with a large number of slicing planes.

### 5.3. Results

- **Spun-trefoil knot.** Figure 5(a) shows the result of rendering the spun trefoil without haloed occlusion cutaways; the translucent effects added in Figure 5(b) do not provide much help in showing the spatial relations. Figure 5(c) is an expensive fully 4D-illuminated volume rendering with 4D occlusions but no expansion of the occlusions using halos. The volumetric halos in Figure 5(d) emphasize the 4D occlusions in the same manner as gaps in a 3D knot-diagram visualization, carving out holes in surface parts that are shadowed by other parts in the projection.
- **Quadratic Fermat.** Figure 6(a) shows a 2nd-order Fermat surface given by $(z_1)^2+(z_2)^2=1$ in $CP(2)$. There are two pinch points (see the white arrows) at which haloed cutaways begin, starting with zero 4D depth difference and monotonically increasing away from the pinch points; this is a particularly clear illustration of the value of having the 4D depth of the halos increasing with the distance from the occlusion curve.
- **Calabi-Yau quartic cross-section.** This manifold, the K3 cross-section, is a relatively more complicated 4th-order manifold in $CP(2)$, with 16 surface patches and numerous intersections in the projected image. Figure 6(b) shows our illustrative visualization result with volumetric halos.
- **Product polynomial.** Figure 7 shows the degree two product polynomial surface in $CP(2)$ described locally by the complex equation $(z_1)^2(z_2)^2=1$. The stereoscopic viewing provides better perception of the spatial relations among different surface patches in the projected volume image, and the depth-dependent volumetric halos further emphasize the occlusions in the projection from the original 4D space to 3D.
- **Twist-spun-trefoil knot.** Figure 8 illustrates another

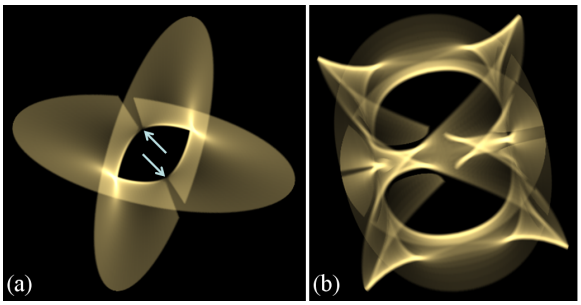


**Figure 6:** *(a) Quadratic Fermat; (b) Calabi-Yau quartic cross-section.*

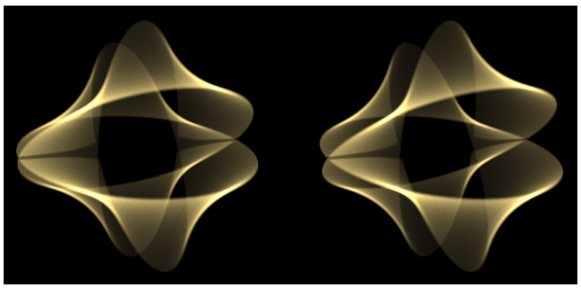

**Figure 7:** *The degree two $CP(2)$ product polynomial: cross-eyed stereo.*

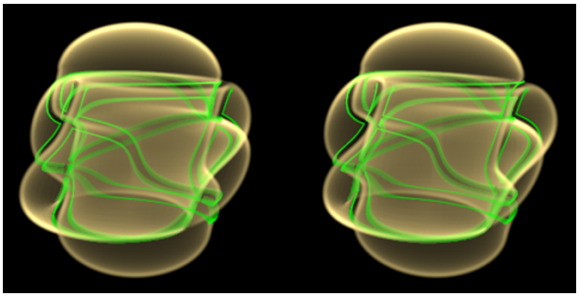

**Figure 8:** *Twist-spun-trefoil knot: cross-eyed stereo and halo boundaries highlighted in green.*

complicated 4D model, based on adding a twist to the trefoil knot used to generate the topological sphere. Even when transparency is combined with the halo cutaways to show the internal structure of the geometry, it is still quite hard to get a clear mental map of the model due to the complexity of the internal intersections. A cross-eyed stereo representation with annotated cutaway curves is shown to assist in extracting the structure.

## 6. Conclusion

In this paper, we generalize the halo visualization methods from 3D graphics to 4D graphics. We develop the notion of illustrative volumetric halos with the goal of more clearly revealing the 4D occlusions of geometric objects that are artifacts of the 4D to 3D projection. We design and implement halo-based occlusion-enhancement methods, and employ GPU-based techniques enabling real-time performance to enhance the viewer's interactive exploration experience. Stereoscopic viewing is supplied to assist in understanding the full structure of 3D volume imaging geometry.

## Acknowledgments

The work described in this paper was supported by a grant from Ministry of Science and Technology of the People's Republic of China under the Singapore-China 9th Joint Research Program (Project No. 2013DFG12900), a grant from Innovation and Technology Commission of Hong Kong (Project No. ITS/138/11), and Singapore MOE Tier-1 Grant (RG 29/11).

## References

[ARS79] APPEL A., ROHLF F., STEIN A.: The haloed line effect for hidden line elimination. In *SIGGRAPH 1979* (1979), pp. 151–157. 1, 2

[Ban90] BANCHOFF T. F.: Beyond the third dimension: Geometry, computer graphics, and higher dimensions. *Scientific American Library* (1990). 2

[Ban92] BANKS D. C.: Interactive manipulation and display of two-dimensional surfaces in four-dimensional space. In *Symposium on Interactive 3D Graphics* (1992), pp. 197–207. 2

[BG07] BRUCKNER S., GRÖLLER M. E.: Enhancing depth-perception with flexible volumetric halos. *IEEE TVCG (Proceedings of IEEE Visualization) 13*, 6 (2007), 1344–1351. 2

[CFHH09] CHU A., FU C.-W., HENG P.-A., HANSON A. J.: GL4D: A GPU-based architecture for interactive 4D visualization. *IEEE TVCG (Proceedings of IEEE Visualization) 15*, 6 (2009), 1587–1594. 1, 2, 5

[CH94] CROSS R. A., HANSON A. J.: Virtual reality performance for virtual geometry. In *Proceedings of IEEE Visualization 1994* (1994), pp. 156–163. 1

[DMNV12] DÍAZ J., MONCLÚS E., NAVAZO I., VÁZQUEZ P.-P.: Adaptive cross-sections of anatomical models. *Computer Graphics Forum (Proceedings of PG) 31*, 7-2 (2012), 2155–2164. 2

[DWE03] DIEPSTRATEN J., WEISKOPF D., ERTL T.: Interactive cutaway illustrations. *Computer Graphics Forum 22*, 3 (2003), 523–532. 2

[EBRI09] EVERTS M. H., BEKKER H., ROERDINK J. B. T. M., ISENBERG T.: Depth-dependent halos: Illustrative rendering of dense line data. *IEEE TVCG (Proceedings of IEEE Visualization) 15*, 6 (2009), 1299–1306. 2, 3

[ER00] EBERT D. S., RHEINGANS P.: Volume illustration: non-photorealistic rendering of volume models. In *Proceedings of IEEE Visualization 2000* (2000), pp. 195–202. 2

[GGSC98] GOOCH A., GOOCH B., SHIRLEY P., COHEN E.: A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH 1998* (1998), pp. 447–452. 2

[HC93] HANSON A. J., CROSS R. A.: Interactive visualization methods for four dimensions. In *Proceedings of IEEE Visualization 1993* (1993), pp. 196–203. 1

[HCV52] HILBERT D., COHN-VOSSEN S.: *Geometry and the Imagination*. Chelsea, New York, 1952. 2

[HGH*10] HUMMEL M., GARTH C., HAMANN B., HAGEN H., JOY K. I.: IRIS: Illustrative rendering for integral surfaces. *IEEE TVCG (Proceedings of IEEE Visualization) 16*, 6 (2010), 1319–1328. 2

[HH92] HANSON A. J., HENG P. A.: Illuminating the fourth dimension. *IEEE Computer Graphics and Applications 12*, 4 (1992), 54–62. 2

[Hol91] HOLLASCH S. R.: Four-space visualization of 4D objects, 1991. Master thesis, Arizona State University. 2

[HZ91] HOFFMANN C. M., ZHOU J.: Some techniques for visualizing surfaces in four-dimensional space. *Computer Aided Design 23*, 1 (1991), 83–91. 2

[IG98] INTERRANTE V., GROSCH C.: Visualizing 3D flow. *IEEE Computer Graphics and Applications 18*, 4 (1998), 49–53. 1, 2

[Ino13] INOUE A.: A symmetric motion picture of the twist-spun trefoil. *Experimental Mathematics 22*, 1 (2013), 15–25. 1

[KLMA10] KARPENKO O. A., LI W., MITRA N. J., AGRAWALA M.: Exploded view diagrams of mathematical surfaces. *IEEE TVCG (Proceedings of IEEE Visualization) 16*, 6 (2010), 1311–1318. 2

[LCD06] LUFT T., COLDITZ C., DEUSSEN O.: Image enhancement by unsharp masking the depth buffer. In *SIGGRAPH 2006* (2006), pp. 1206–1213. 2

[LRA*07] LI W., RITTER L., AGRAWALA M., CURLESS B., SALESIN D.: Interactive cutaway illustrations of complex 3D models. *SIGGRAPH 2007 26*, 3 (2007). 2

[Nol67] NOLL A. M.: A computer technique for displaying N-dimensional hyperobjects. *Communication ACM 10*, 8 (1967), 469–473. 2

[SMSE00] SCHUSSMAN G., MA K.-L., SCHISSEL D., EVANS T.: Visualizing DIII-D tokamak magnetic field lines. In *Proceedings of IEEE Visualization 2000* (2000), pp. 501–504. 2