Adaptive Cutaways for Comprehensible Rendering of Polygonal Scenes

Michael Burns
Princeton University

Adam Finkelstein
Princeton University

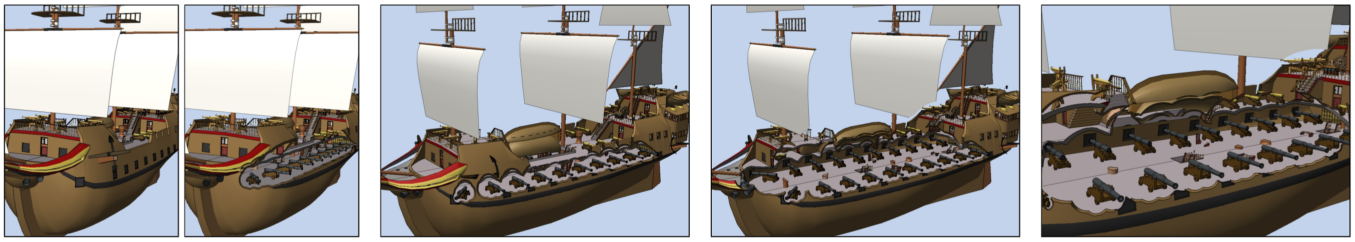


Figure 1: *From left to right, the hull of a galleon is cut away to reveal cannons on the interior. Changing camera updates the cutaway shape, exemplified by the leftmost exposed cannon. A further row of cannons is exposed, and the camera is zoomed in.*

Abstract

In 3D renderings of complex scenes, objects of interest may be occluded by those of secondary importance. Cutaway renderings address this problem by omitting portions of secondary objects so as to expose the objects of interest. This paper introduces a method for generating cutaway renderings of polygonal scenes at interactive frame rates, using illustrative and non-photorealistic rendering cues to expose objects of interest in the context of surrounding objects. We describe a method for creating a view-dependent cutaway shape along with modifications to the polygonal rendering pipeline to create cutaway renderings. Applications for this technique include architectural modeling, path planning, and computer games.

Keywords: Cutaway diagram, NPR, visibility, distance transform

1 Introduction

With advances in 3D modeling, processing, and rendering capabilities, challenges brought by model complexity increasingly involve comprehension. Complex scenes may contain hundreds or thousands of objects, many in close proximity to each other and often nested within other objects. Challenges include visual clutter, navigation, and occlusion. This paper addresses the occlusion challenge by offering a method for exposing hidden objects of interest while preserving contextual information of the scene.

The problem of occlusion occurs in many interactive scenarios with changing camera and dynamic scene structure. In modeling complex scenes like buildings with interiors, it is often difficult to position and view objects in building interiors, as objects are easily occluded by walls and other objects. In VR and gaming, camera views that are far away from a character are limitedly effective because once a character enters or moves behind any type of opaque structure, they are no longer visible.

One strategy for revealing *objects of interest* occluded by objects of secondary importance (*secondary objects*) is to simply omit the latter. Unfortunately, this strategy also removes information about objects' placement, or context, in the scene. Artists have addressed this problem in the past by creating *cutaway* illustrations that selectively omit portions of secondary objects in the area in front of the objects of interest, relative to the viewpoint. Such illustrations give the impression that secondary objects have been cut by knife so as to remove occluding material and expose objects of interest.

Our goal is to create cutaway renderings of complex, dynamic polygonal scenes under changing viewpoint at interactive frame rates, allowing one to effectively visualize objects of interest in the context of their surroundings. Cutaways that automatically expose individual pieces of furniture, whole room interiors, or characters moving through buildings could ease problems with camera placement and provide enriched visualization experiences.

There are several challenges associated with creating cutaway illustrations. One challenge is in determining the regions of 3D space that occlude objects of interest in a scene. This region may be represented by a *cutaway surface* that defines the boundary between occluding and non-occluding regions. Previous work has described image-based cutaway surface generation techniques using signal processing algorithms or geometry-based techniques using CSG algorithms, but have suffered from non-interactivity in either changing viewpoint or dynamic object geometry.

Another challenge is the visualization of *cut surfaces*, which lie at intersections of the cutaway surface with interiors of secondary objects that are cut away. Put another way, cut surfaces are interior surfaces of solid objects that are made visible by the cutaway action. Volumetric rendering naturally renders interior materials of solid objects that have been exposed by the cutaway so no special care needs to be taken in cutaways of volumetric material. Previous efforts in polygonal rendering have created explicit cutaway surface geometry that can be rendered with CSG stencil buffer techniques, which can be expensive for changing cutaway shape.

To address these challenges, we introduce a flexible image-space cutaway surface generation algorithm based on jump flooding [Rong and Tan 2006] to generate a depth-image representation of the cutaway surface at interactive frame rates. We then introduce augmentations to the polygonal rendering pipeline to remove occluding material and to render exposed cut surfaces. We also introduce non-photorealistic rendering techniques to provide additional context information for secondary objects. We present results that show how these techniques can be used to effectively expose objects of interest along with the contextual placement and structure of secondary objects in complex, dynamic 3D scenes.

2 Related Work

Previous work to address this visibility challenge may be broadly separated into volumetric and polygonal rendering methods.

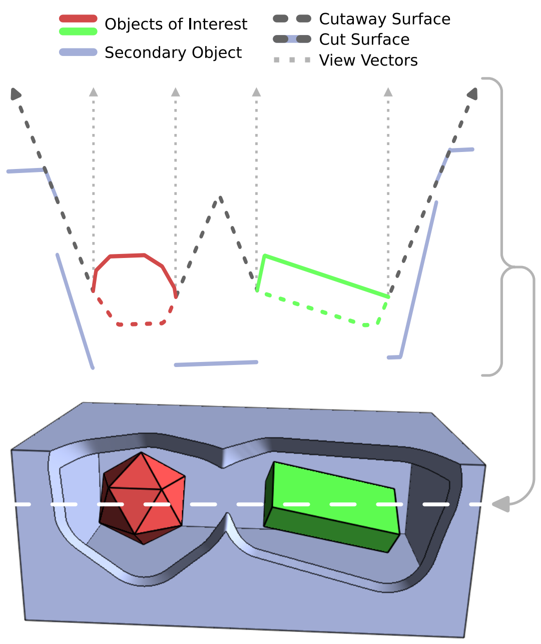


Figure 2: *A cutaway illustration is shown beneath a plot of depth values for a single row of pixels (white dotted line). Material from the secondary object (blue) is removed to expose the objects of interest (green and red), which define the shape of the cutaway. Solid, dashed, and dotted lines in the plot correspond to visible pixels, cutaway surface, and camera rays, respectively*

Volumetric rendering. Context-preserving volume rendering [Bruckner et al. 2006], opacity peeling [Rezk-Salama and Kolb 2006], and the Clearview system [Krüger et al. 2006], generate visualizations by ghosting and fading outer layers of material to reveal inner structures, sometimes incorporating user-specification of regions of interest. User-defined regions have been exposed in peel-away [McGuffin et al. 2003; Correa et al. 2006] and exploded visualizations [Bruckner and Gröller 2006]. Contextual cutaway views have been used in medical imaging applications, to expose an ultrasound plane embedded within a volume [Burns et al. 2007].

Importance-driven volume rendering [Viola et al. 2004] introduces fully automatic object and view-dependent cutaway surface generation by using signal processing on the depth buffer, and introduces multiple rendering styles for the material in the cutaway region, experimenting with transparency, screening, and other density-reducing effects. Our work takes a similar approach with respect to the cutaway representation but abandons the use of Borgefors' chamfer distance transform approximation [Borgefors 1986] for a more flexible jump flooding algorithm [Rong and Tan 2006], which allows greater explicit and adaptive control over the cutaway shape. Jump flooding was introduced as a parallelizable algorithm to create 2D distance transforms and Voronoi diagrams.

Polygonal rendering. Early cutaway work [Feiner and Seligmann 1992] used z-buffer and image processing techniques to create images where occluding surfaces in front of objects of interest were replaced by semi-transparent representations. Nested 3D objects have been displayed using simple static geometric cutouts [Diepstraten et al. 2003] defined by the intersection of two or more half-spaces and breakaway views defined by inflated convex hulls of objects. Predefined shape cutaways [Liang et al. 2005], where the cut is influenced by a user-specified shape, have been used in medical imaging applications to mimic the cuts a surgeon might make in the course of an operation. Other techniques incorporate user-specification of the cutting surface [Coffin and Hollerer 2006] to create 3D geometric cutting volumes and use CSG rendering techniques to render cut surfaces. Recent work on interactive illustrative cutaways [Li et al. 2007] introduces a flexible and robust illustration system for static scenes based on cutaway views that depends heavily on user-specification of geometry types for all objects in a scene and, due to high computational overhead, is non-interactive for changing cuts and moving to arbitrary viewpoints.

3 Object-Defined Cutaways

Previous work on view-dependent cutaways has been unable to achieve interactive frame rates for dynamic scenes under changing viewpoint. In this scenario, occlusion of objects may change from one frame to the next, requiring the cutaway action to be updated per-frame. In order to minimize computation per frame, we have chosen a cutaway style that avoids both expensive analysis of high-level scene structure and extensive processing of secondary objects, instead focusing processing time on a cutaway shape determined only by the objects of interest. We have also chosen an image-based cutaway representation (Section 3.1) and computation that scales well with scene and model complexity and adds only a small overhead to the overall scene rendering time. Such limited processing results in a style that may violate traditional cutaway principles, particularly in the shape of cuts in objects of secondary importance, but still facilitates cutaway illustrations that exhibit several fundamentally desirable properties (Section 3.2).

3.1 Depth Image Cutaway Representation

At the heart of our cutaway rendering algorithm is the depth image representation of the cutaway surface. Previous work [Viola et al. 2004] uses a chamfer distance transform approximation [Borgefors 1986] to create a depth image of a basic cutaway surface. This requires rendering the back hulls of objects of interest into a depth buffer, reading this buffer to the CPU to be processed by the chamfer algorithm, then writing the buffer back to video memory.

The chamfer distance transform on a binary image B approximates a Euclidean 2D distance transform, which determines the minimum distance d at each pixel $p = (x, y)$ to an active pixel in B:

$$d(p) = \min_{q \in B} \|q - p\| \tag{1}$$

Note that this is equivalent to maximizing the following function, where d_{max} is a maximum value for d, and $d(p) = d_{max}$ would indicate that p corresponds directly to an active pixel in B:

$$d(p) = \max_{q \in B} (d_{max} - \|q - p\|)$$

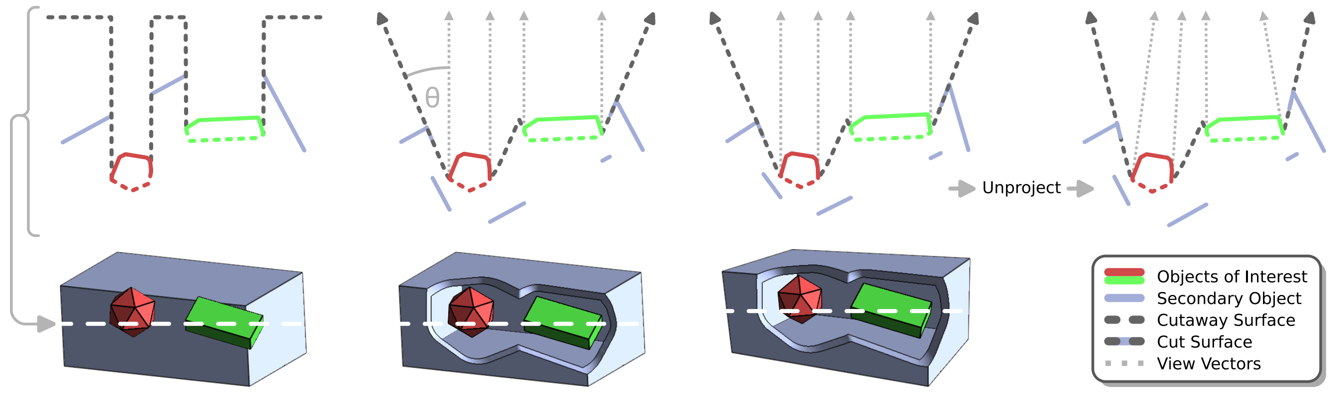
By using a floating point input image I initialized with variable depth values $z(p)$ instead of d_{max}, and scaling the chamfer kernel by m, the chamfer algorithm maximizes the following function:

$$d(p) = \max_{q \in I} (z(p) - m\|q - p\|) \tag{2}$$

The surface defined by this function resembles the effect of an infinitely wide drill bit with slope m to the image plane passing over each pixel p and drilling down along the z-axis to the depth $z(p)$. This surface can also be thought of as the rear hull of cones with slope m to the image plane anchored along the z-axis and positioned over each pixel at the depth $z(p)$. A z-aligned cone with slope m to the image plane makes an angle $\theta = \tan m$ to the z-axis.

3.2 Basic Cutaway Surface

The surface defined by Equation 2 can represent a cutaway, by using an input image I with depth values from the objects of interest. The image I is initialized with the value of the near-plane depth (zero in OpenGL), and the rear hulls of objects of interest are rendered into the image so they "punch into" the near-plane, as shown in Figure 3(a). Calculating the surface with $m = \tan^{-1}\theta$ results in a cutaway surface that makes an angle θ to the viewing vector, in an orthographic projection (Figure 3(b)). The cutaway action discards



(a) Orthographic Initialization (b) Orthographic Cutaway (c) Perspective Cutaway

Figure 3: *Depth plots with an orthographic camera are shown immediately after initialization (a) and after calculation of the cutaway surface (b), which makes an angle θ to the view vector. With a perspective camera (c), calculation from post-projection input depths creates a similar result. Un-projecting the depth values to world space shows that basic qualities of the cutaway shape are maintained.*

portions of secondary objects with depth less than that of the surface (in front of the surface). Such a cutaway surface exhibits the following useful properties:

- The surface is view dependent so portions of secondary objects that occlude objects of interest can be detected, and discarded.
- Portions of secondary objects can be discarded in various amounts depending on their z-distance from the object of interest, allowing viewing of nested structures.
- The amount of material discarded can be controlled by the angle θ of the cutaway as desired, providing a balance between the amount of material removed and the ability to locate the objects within nested structures.
- The shape of the cutaway hugs the silhouettes of the objects of interest, reducing the amount of material that has to be removed in order to expose an object.
- Multiple objects of interest can define a single cutaway surface, with the cutaway surface exposing all objects of interest jointly.

In a perspective projection, the modified distance transform of post-projection depth values generates a surface with slightly different, but still useful, geometric properties. Since linear surfaces are preserved in the perspective transformation, the surface is still conical with linear edges but instead of the cone axes being aligned along the z-axis, they are aligned along the viewing rays, as shown in Figure 3(c). The precise angle of each cone is no longer constant, since the slope m of a line can change depending on its depth and projected position during the perspective transform. However, the surface generated still exhibits all of the properties enumerated above, so the transform remains valuable for creating a cutaway surface in perspective projection scenarios.

4 Rendering Methods

In order to create view-dependent cutaway renderings, we use a multi-pass rendering pipeline that consists of three main steps. In the first step, the rear hulls of objects of interest are rendered into a depth buffer, which is then processed to create a depth image representation of a cutaway surface. The basic cutaway definition has been described in Section 3, but further refinements, as well as computation details, are described in Section 5. In the second step, the secondary objects of the scene are rendered and clipped against the cutaway depth image, and the objects of interest are rendered completely (Section 4.1). In a final step, lines are composited to give additional information about the shape of cut surfaces (Section 4.2) and structures of partially clipped objects (Section 4.3).

4.1 Cutting Polygons

For a given cutaway surface depth image, we would like to render only portions of secondary objects behind the cutaway surface. Since current graphics hardware does not support more than one depth buffer test, we implement this in a fragment shader by comparing a fragment's depth with the cutaway depth value at the fragment's location in screen space, discarding fragments appropriately. This clipping is shown in Figure 4(a) and results in a hollow appearance to solid objects, as no additional fragments are rendered to represent the cut surfaces.

This problem can be addressed by the following simple observation. For a scene composed of non-intersecting water-tight models, partial clipping of models always exposes their interiors. Pixels from these interiors can be detected as coming from back-facing triangles of the model being clipped. Thus, in order to give the appearance of rendered cut surfaces, one can simply shade fragments from back-facing triangles using normals derived from the cutaway shape, as shown in Figure 4(b). This gives the appearance of solid objects being carved away along the cutaway surface and aids in the perception of the cutaway shape.

Only secondary objects are clipped by the cutaway, and objects of interest are rendered without clipping. The cutaway nature dictates that secondary objects never occlude objects of interest, so objects of interest may be rendered in a second pass.

4.2 Cut Surface Outline

The rendering of cut surfaces gives the impression of solid objects being carved away by the cutaway surface. This carving creates 3D ridges on partially clipped secondary objects that are distinct from ridges defined by the object geometry alone. In a non-photorealistic rendering scenario, one may render lines to accentuate ridges and creases on object geometry. In order to complete these renderings, one may render the ridges generated by the carving action.

These ridges are located on image-space boundaries of the cut surfaces rendered in Section 4.1. Cut surfaces can be outlined by recording fragments from back-facing triangles in a separate binary channel and processing this image on the GPU to highlight or mark pixels on the binary image boundaries. Figure 4(c) shows how these outlines correspond to the ridges formed by the clipping, and aid visual perception of the cut surface locations.

This image-space operation can fail to work in situations where the clipped object has little separation between front and back-facing surfaces and the cutaway surface is very steep relative to the camera. In order to address this problem, back-facing fragments are clipped to a cutaway surface that is slightly narrower,

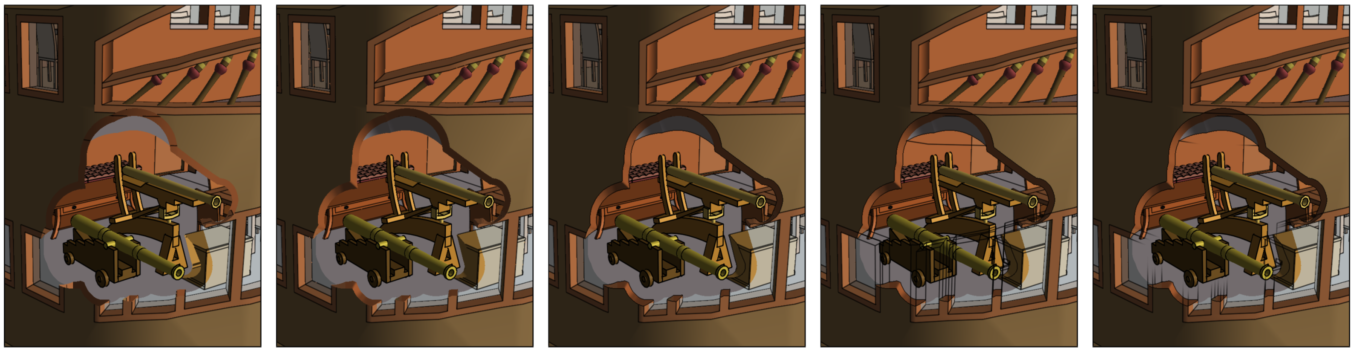

(a) Clipping only (b) With cut surface (c) With outlines (d) With ghost lines (e) With faded ghost lines

Figure 4: *Rendering effects: Walls of the cut secondary shape appear hollow (a), so a "cut surface" is rendered (b) and outlined (c) to give the walls a solid appearance. "Ghost lines" indicating removed structures are rendered (d) and faded (e) to reduce visual clutter.*

such that at least one back-facing fragment is exposed where a surface is clipped. This surface is created by a dilation operation on the cutaway depth image that finds the minimum value in a 1-pixel neighborhood. The resulting dilated cutaway depth image is used to clip back-facing fragments when detecting cut surface outlines as well as when rendering cut surfaces in Section 4.1 for consistency.

4.3 Ghost Lines

Feature lines such as contours or creases help show the structure of an object. Rendering such lines without the corresponding polygons often provides an impression of the structure, while largely preventing the object from occluding objects behind it. When portions of secondary objects are carved away, we can allow some of these *ghost lines* to extend into the cutaway region, to show how the structure of the carved object exists towards the interior of the cutaway. Lines are drawn up to a user-specified distance in front of the cutaway surface in order to prevent visual clutter and are faded out to avoid abrupt termination, as shown in Figure 4(e).

5 Locally Controlled Cutaway Surface

A fast computation method (Section 5.1) can be used to provide local control over the cutaway surface, to address several visual shortcomings of the basic cutaway surface (Sections 5.2-5.5).

5.1 Cutaway Surface Computation

Jump flooding, as presented by Rong and Tan [2006], is a GPU-based version of a parallelizable algorithm for calculating exact 2D distance transforms by minimizing Equation 1. This algorithm operates in $O(n^2 \log n)$ time, by performing $\log n$ passes over an n-by-n image. By replacing Equation 1 with Equation 2 in the jump flooding algorithm, we are able to generate the basic cutaway surfaces described in Section 3. As described in Section 6, this operates significantly faster than the Chamfer approximation, while producing a surface free of approximation artifacts.

Computation begins by rendering the objects of interest into a buffer, encoding the screen-space coordinates and depth at each pixel. The "greater than" depth function is used so fragments furthest from the camera, comprising the rear hulls, are stored. For an n-by-n image, the algorithm executes $\log n$ iterations over the buffer, with a single per-iteration parameter $k = n/2^i$ for iteration i.

As described in Section 3 a cutaway surface can be defined as the maximum of a cutaway function $C(p,q)$ at p for all q in the image. In the case of the basic cutaway surface (Equation 2) we have:

$$C(p,q) = z(p) - m\|q - p\|$$

For each iteration and pixel p, the stored depth and coordinates at p, which are the maximum found depth and its coordinate location, is compared to $C(p,q)$ evaluated for the 8 candidate points $q \in \{(p_x + \{k,0,-k\}, p_y + \{k,0,-k\}) - p\}$. The updated maximum depth and its coordinate location are stored at p for the next iteration. At the completion of $\log n$ iterations, the depth at p is the maximum value of $C(p,q)$ for all q over the image. Evaluating $C(p,q)$ exactly each time avoids approximation artifacts and allows definition of the cutaway function per pixel, affording local control.

5.2 Perspective Compensation

In a perspective projection, the cutaway surface performs reasonably well, but because the slope m to the image plane is applied in screen space, a fixed slope m can cause cutaways of very differ-

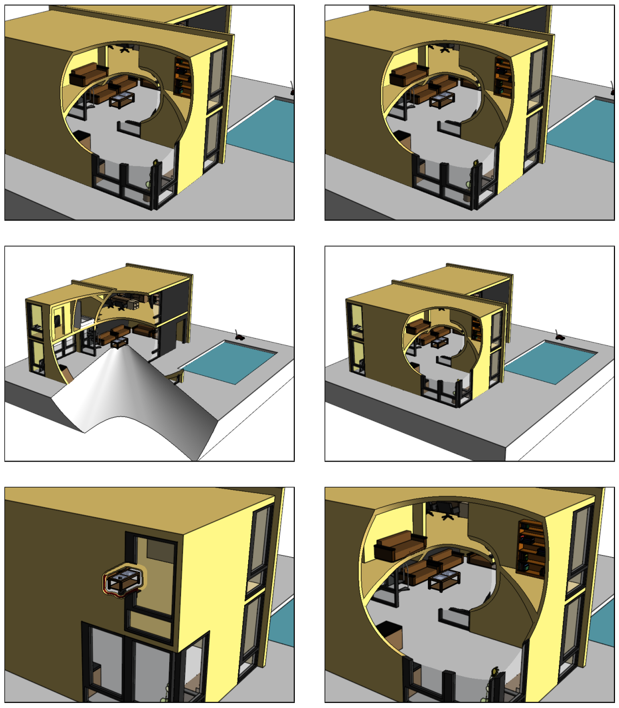
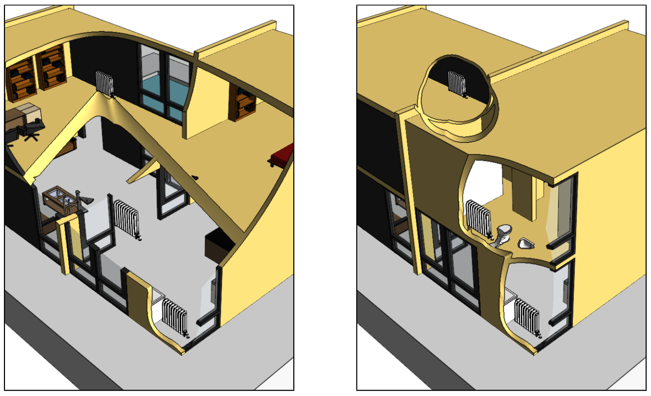

Fixed angle Perspective compensation

Figure 5: *Perspective compensation. Left column: In perspective, it is difficult to choose a single cutaway angle that works well at different distances. Top to bottom: An angle appropriate for a middle distance appears too wide from far away, and too narrow from close up. Right column: With perspective compensation (Section 5.2), the angle chosen for the upper image is appropriate in the other views.*



(a) Global angles (b) Per-object angles

Figure 6: *Local control of cutaway angle. Using a single wide cutaway angle (a) exposes the three radiators of interest in the house, but discards too much secondary material, resulting in a floating radiator in the center of the image. Using different angles for each radiator (b) results in a more effective illustration. (Section 5.3)*

ent widths (angles) in model space, depending on the depth of the anchor point p. This effect is shown in Figure 5, where the cutaway becomes very wide when zoomed out and very narrow when zoomed in. The perspective error can be drastically reduced by applying a scaling factor to the slope, $-(PM_{sz}+z(p))$ where PM_{sz} is the z-scaling factor of the projection matrix, which is defined under OpenGL as $PM_{sz} = (z_{near}+z_{far})\,/\,(z_{near}-z_{far})$. This scaling factor is accurate for the projection of a line passing through the z-axis, and the error of the approximation is larger for lines anchored at p further away from the screen center. Thus we have:

$$m(p) = \frac{-(PM_{sz}+z(p))}{\tan\theta} \tag{3}$$

5.3 Local Angles

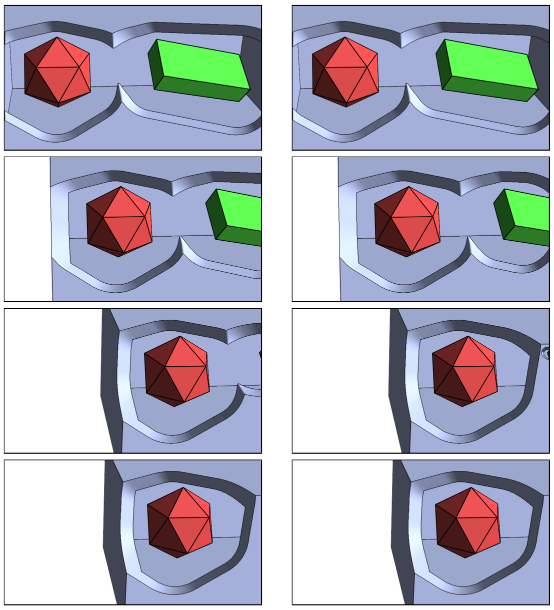
The ability to define the cutaway function per pixel also allows us to define the cutaway angle for each pixel. In practice, it is most useful to define a cutaway angle per object or group of objects of interest and use this information to define a cutaway angle per pixel. Figure 6 shows this effect with different angles for each radiator.

5.4 Edge Compression

A limitation of this image-based approach is that objects of interest that are offscreen do not contribute to the onscreen cutaway surface. This is acceptable for single images as carving secondary objects to reveal objects of interest that are not on screen makes little sense. However, when objects move on or off screen in interactive or animated scenarios, a noticeable discontinuity can occur where objects are suddenly carved away, as shown in Figure 7.

This could be addressed with a cutaway buffer much larger than the screen buffer, so objects outside the rendered screen could still contribute to the cutaway surface, but this would lead to an undesirable increase in processing time and memory requirements. Given local control over the cutaway function, this problem can be mitigated by adjusting the effective angles such that pixels near the edge of the screen have a cutaway with a very narrow angle and pixels sufficiently away from the edge of the screen use their normal angle. Given screen space coordinates for p in the range $(-1,1)$ and a size of the screen edge region r, this can be accomplished by defining:

$$\theta_e(p) = \theta(p)\,\underset{[0...1]}{\text{clamp}}\left(\frac{1-|p_x|}{r}\right)\underset{[0...1]}{\text{clamp}}\left(\frac{1-|p_y|}{r}\right)$$



(a) Uniform cutaway (b) Edge compression

Figure 7: *Edge compression. Top to bottom: Camera pans left. When the green object of interest exits the screen, it normally causes a temporal discontinuity in the cutaway shape (a). With edge compression (Section 5.4), the cutaway shape becomes progressively smaller towards the edge of the screen, mitigating the temporal discontinuity and preventing visual pop.*

We have found that $r = 0.1$ provides a smooth and balanced transition for objects entering and exiting the screen.

5.5 Directional Constraints

Figure 8(a) shows an undesired artifact of cutaways from objects positioned on a floor, where the floor exhibits a distractingly large cut surface. This artifact happens on any large and flat surface that enters the cutaway interior nearly parallel to the edge of the cutaway, as the surface is only slightly carved away. This artifact occurs for common views when objects of interest are situated on top of large secondary objects, such as the floor.

This can be addressed by investigating an angle $\phi(p)$, the maximum angle of cutaway to avoid clipping a surface with normal $\hat{n}$. With the (global) normal of the floor direction $\hat{n}$, and the normalized view vector $\hat{v}(p)$:

$$\phi(p) = \frac{\pi}{2} - \arccos\left(\hat{v}(p)\cdot\hat{n}\right)$$

Based on this angle, which may be negative when the floor is viewed from below, a suitable cutaway angle can be calculated to minimize undesirable carving of the floor. The cutaway angle θ can be clamped to ϕ for camera positions above the floor (positive ϕ) and quickly transition from zero to θ once the camera passes below the floor plane (negative ϕ). Thus we have:

$$\theta_f(p) = \underset{[0...\phi(p)]}{\text{clamp}}\,\theta_e(p) + \theta_e(p)\,\underset{[0...\frac{\pi}{18}]}{\text{smoothstep}}\left(-\phi(p)\right)$$

Application of this angle to the cutaway directly would shrink the cutaway in all screen-space directions as the floor becomes more perpendicular to the camera. However, the cutaway should only be shrunk in screen-space directions where it may cut into the floor. This can be addressed by using θ_f for directions of the cutaway opposite the projected screen direction of the floor normal $\hat{n}_{proj}$ and θ_e for other directions. This is accomplished by establishing α as the coherence between the screen space direction of q to p and the projected floor normal and using α to interpolate between cutaway

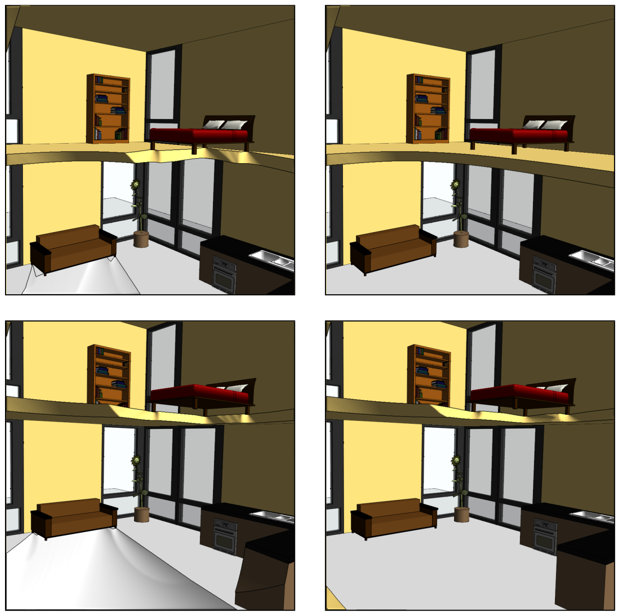

(a) Cut floor (b) Directional constraint

Figure 8: *Directional constraint. (a) Objects of interest directly on the floor can cause a cutaway surface to intersect the floor, even when viewed from above. (b) With a directional constraint, the cutaway surface avoids the floor. (Section 5.5)*

angles θ_e and θ_f. Thus the final cutaway function, where $m_e(p)$ and $m_f(p)$ are generated by Equation 3 from $\theta_e(p)$ and $\theta_f(p)$:

$$\alpha = \underset{[0...1]}{\text{clamp}} \left(\frac{(q-p)}{\|q-p\|} \cdot \hat{n}_{proj} \right)$$

$$C(q,p) = z(p) - ((1-\alpha)\, m_e(p) + \alpha m_f(p))\, \|p-q\|$$

Figure 8(b) shows the resulting effect where the cutaway no longer creates large distracting patches of cut surface and adjusts itself with respect to the floor and camera directions.

6 Results

All models shown in figures in this paper render at interactive frame rates and may be seen in motion in the accompanying video. The performance numbers in Table 1 show the efficiency of our algorithm for models of various sizes. In the middle row we see the efficiency of the jump flood algorithm for the cutaway surface computation. Viola et al. [2004] calculated a similar kind of cutaway surface for volumetric data using the chamfer distance transform approximation, so we include this algorithm for comparison. Our tests indicate a significant performance advantage for the GPU-capable jump flood algorithm. Overall, this algorithm does not impose a substantial performance overhead over standard polygonal rendering. The performance numbers were measured on an NVIDIA GeForce 8800 GTS 512 with a framebuffer size of 640 × 480 and a cutaway buffer size of 512 × 512.

Figure 9 shows a series of pictures from the exterior of an office building with rooms selected for viewing. The goal of the images is to show the location and nearby structures of the rooms within the context of the building. Unlike in previous images shown, the cutaway surface is defined not by the objects of interest, the furniture in each room, nor the walls of the room itself, but by an invisible shape that is only rendered to the cutaway surface. In the case of these two rooms, a simple box that spans almost the extent of each room is sufficient to create this effect, along with rendering the enclosed furniture as objects of interest.

	House	Galleon	Building
Triangles (K)	76	301	530
Lines (K)	45	92	286
Chamfer (fps)	10	10	10
Jump Flood (fps)	60	60	60
Model (fps)	46	17	11
With Cutaway (fps)	41	16	10

Table 1: *Model sizes and frame rates for the house, galleon, and building models shown in Figures 10, 1, and 9, respectively. Chamfer denotes the frame rate for approximating the cutaway surface using a 5 × 5 chamfer kernel, whereas the Jump Flood algorithm can compute the surface more accurately and quickly, and permits local adjustments described in Section 5. Rendering the full model is generally the bottleneck; calculation of the cutaway surface and additional rendering effects impose a relatively small overhead.*

Our algorithm has a low cost per-frame and does not rely on positional structural information of the scene, making it suitable for use in both off-line and interactive animations. Figure 10 shows several frames of an animation with a radio-controlled car driving into and around a house, where the RC car and the kitchen counter are the objects of interest. When multiple objects of interest are near each other, they both contribute to the cutaway surface in that region. We see as the car passes near and in front of the counter that it creates a surface which reflects both the shape of the counter and the shape and position of the car. When the car is sufficiently far away from the counter, the cutaway surfaces are effectively separate. Note that objects of interest never cause each other to be cut; rather they both contribute to the cutaway surface. Thus, they can be rendered in one pass using regular depth-testing amongst themselves, and may occlude each other.

Because the cutaway surface tends to be smooth, it is often sufficient to use a cutaway buffer with resolution lower than that of the screen, since a linearly interpolated low-resolution depth-image of the cutaway surface closely approximates the high-resolution equivalent. We typically use a buffer size of 512 × 512 at screen sizes up to 1024 × 768 and find this to be sufficient. As it is an image-space algorithm, aliasing can and does occur, especially at low resolutions, but in most cases is not significantly noticeable.

7 Conclusions, Limitations and Future Work

This paper introduces a method for creating cutaway illustrations for revealing objects of interest in complex polygonal scenes. The cutaway shape is dynamic, smoothly responding to animated scenery and changes in view. We describe methods for creating the cutaway shape as well as several non-photorealistic rendering effects at interactive frame rates. The methods described in this paper have several limitations that could inspire future research:

Framebuffer/resolution. The in-frame nature of these algorithms lead to several related artifacts. First, as mentioned in Section 5.4, only onscreen objects of interest contribute to the cutaway shape, necessitating the edge compression method to avoid temporal artifacts. Second, the calculation of the cutaway shape occurs at a specific resolution, so objects of interest with very thin components can induce aliasing in the cutaway shape. Ideally, such objects would be replaced with a surrogate invisible object with an appropriate level of detail (LOD) as in Figure 9. We believe that a fruitful area for future work would be automatically finding surrogate shapes of an appropriate LOD, perhaps using methods inspired by those of Li et al. [2007].

Shape of cutout. The shape of the cutout surface is dictated automatically by the placement of the objects of interest in the scene (relative to the camera) and a user-given angle parameter (possibly per-object). Some applications might benefit from greater user-

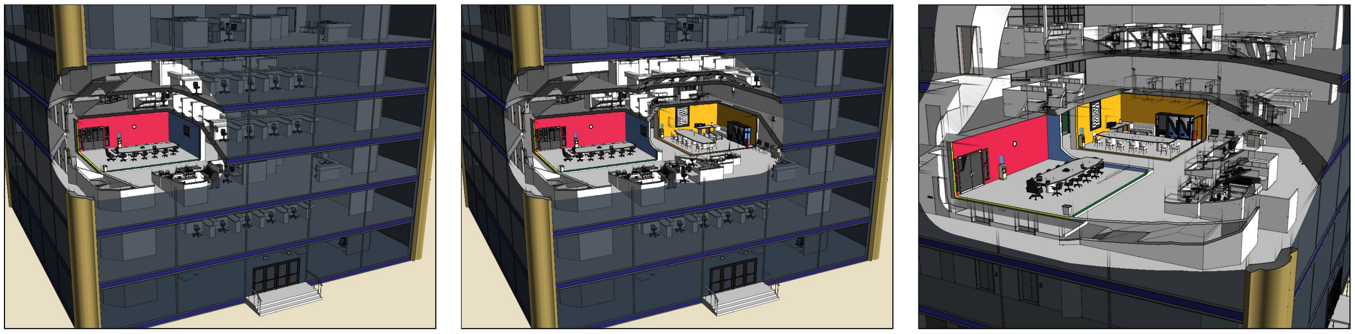


Figure 9: *Stand-in object of interest. Left to right: A simple box, invisible, denotes a boardroom in an office building as the object of interest. A box representing a break room (yellow) is added. A different camera position and a wider cutaway angle expose the upper floors.*

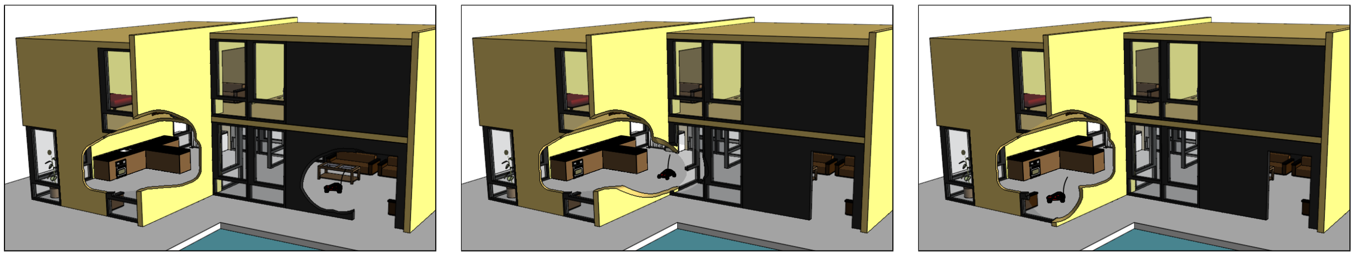


Figure 10: *Animated objects of interest: The radio-controlled car approaches a kitchen counter. The cutaway shape blends smoothly as the car moves, with the two distinct cutaway shapes in the leftmost image merging as the car enters the cutaway region of the counter.*

control of the cutout shape, for example the ability to change the "eccentricity" of the cutout or to bias the cutout direction so that it is not heading as directly towards the camera.

Emphasis. While the technique described here can reveal objects of interest, the resulting imagery remains as visually cluttered as the model from which it is rendered. This technique would be nicely complemented by a method for placing visual emphasis on the objects of interest. The rendering method of Cole et al. [2006] draws the attention of the viewer based on a desired per-pixel target emphasis for lines and surfaces in the scene, typically by evaluating distance to a single point of emphasis. It would be interesting to examine ways of creating emphasis values based on a set of spatially distributed objects of interest.

Acknowledgements

We would like to thank the paper reviewers, the Tiggraph gang, and Wilmot Li for many helpful suggestions. Many of the models shown herein are from the Google 3D Warehouse. This work was sponsored in part by the NSF grant IIS-0511965.

References

BORGEFORS, G. 1986. Distance transformations in digital images. *Comp. Vision Graph. Image Process. 34*, 3, 344–371.

BRUCKNER, S., AND GRÖLLER, M. E. 2006. Exploded views for volume data. *IEEE TVCG 12*, 5, 1077–1084.

BRUCKNER, S., GRIMM, S., KANITSAR, A., AND GRÖLLER, M. E. 2006. Illustrative context-preserving exploration of volume data. *IEEE TVCG 12*, 6, 1559–1569.

BURNS, M., HAIDACHER, M., WEIN, W., VIOLA, I., AND GRÖLLER, E. 2007. Feature emphasis and contextual cutaways for multimodal medical visualization. In *Eurographics / IEEE VGTC Symposium on Visualization 2007*, 275–282.

COFFIN, C., AND HOLLERER, T. 2006. Interactive perspective cut-away views for general 3D scenes. In *Proc. of 3DUI 2006*, 25–28.

COLE, F., DECARLO, D., FINKELSTEIN, A., KIN, K., MORLEY, K., AND SANTELLA, A. 2006. Directing gaze in 3D models with stylized focus. In *Proc. of EGSR 2006*, 377–387.

CORREA, C., SILVER, D., AND CHEN, M. 2006. Feature aligned volume manipulation for illustration and visualization. *IEEE TVCG 12*, 5, 1069–1076.

DIEPSTRATEN, J., WEISKOPF, D., AND ERTL, T. 2003. Interactive cutaway illustrations. In *Proc. of Eurographics 2003*, 523–532.

FEINER, S., AND SELIGMANN, D. D. 1992. Cutaways and ghosting: satisfying visibility constraints in dynamic 3D illustrations. *The Visual Computer 8*, 5&6, 292–302.

KRÜGER, J., SCHNEIDER, J., AND WESTERMANN, R. 2006. Clearview: An interactive context preserving hotspot visualization technique. *IEEE TVCG 12*, 5, 941–948.

LI, W., RITTER, L., AGRAWALA, M., CURLESS, B., AND SALESIN, D. 2007. Interactive cutaway illustrations of complex 3D models. In *Proc. of ACM SIGGRAPH 2007*, 31.

LIANG, R., CLAPWORTHY, G., KROKOS, M., AND MAYORAL, R. 2005. Real-time predefined shape cutaway with parametric boundaries. *Computer Graphics, Imaging and Vision: New Trends, 2005. International Conference on*, 227–231.

MCGUFFIN, M. J., TANCAU, L., AND BALAKRISHNAN, R. 2003. Using deformations for browsing volumetric data. In *Proc. of IEEE Visualization 2003*, 53.

REZK-SALAMA, C., AND KOLB, A. 2006. Opacity peeling for direct volume rendering. *Comp. Graph. Forum 25*, 3, 597–606.

RONG, G., AND TAN, T.-S. 2006. Jump flooding in GPU with applications to voronoi diagram and distance transform. In *Proc. of I3D 2006*, 109–116.

VIOLA, I., KANITSAR, A., AND GRÖLLER, M. E. 2004. Importance-driven volume rendering. In *Proc. of IEEE Visualization 2004*, 139–145.