Smart transparency for illustrative visualization of complex flow surfaces

Robert Carnecky, Raphael Fuchs, Stephanie Mehl, Yun Jang, and Ronald Peikert

Abstract—The perception of transparency and the underlying neural mechanisms have been subject to extensive research in the cognitive sciences. However, we have yet to develop visualization techniques that optimally convey the inner structure of complex transparent shapes. In this paper we apply the findings of perception research to develop a novel illustrative rendering method that enhances surface transparency non-locally. Rendering of transparent geometry is computationally expensive since many optimizations, such as visibility culling, are not applicable and fragments have to be sorted by depth for correct blending. In order to overcome these difficulties efficiently, we propose the illustration buffer. This novel data structure combines the ideas of the A- and G-buffers to store a list of all surface layers for each pixel. A set of local and non-local operators is then used to process these depth-lists to generate the final image. Our technique is interactive on current graphics hardware and is only limited by the available graphics memory. Based on this framework we present an efficient algorithm for a non-local transparency enhancement that creates expressive renderings of transparent surfaces. A controlled quantitative double blind user study shows that the presented approach improves the understanding of complex transparent surfaces significantly.

Index Terms—illustrative rendering, transparency, flow visualization, integral surface, user study, diffusion, a-buffer, illustration buffer, perception

1 Introduction

The shape of a complex surface can be very hard to understand from a single image. The difficulties arise from a wide range of sources: a rendering on the screen does not have binocular depth cues, the lighting is very different from what we are used to, and the often unfamiliar shape does not allow the human visual system (HVS) to apply context knowledge as it often does for real-world objects. Examples of such complex surfaces include integral surfaces, which are an important tool to visualize the behavior of time-dependent flows, and which can contain complex twists and self-intersections. Other examples are Lagrangian coherent structures as used in flow visualization, which are often non-orientable and non-manifold, or nested technical designs for industrial prototypes created in computer aided design (CAD) applications.

For all these examples it is critical to provide users with the capability to see through all surface layers to reveal interior parts of the object. If users need to understand the whole object at once, opaque rendering is not an option. Cutting away parts of the object can be a good approach, however, for many geometries the important structures are layered in a way that a cutaway would remove important information.

Even though transparency can show more information about the object, transparent surfaces are generally more difficult to understand. The aim of this work is to develop a method for assigning transparency in illustrations that improves the understanding of transparent surfaces. In fact, this approach has been adopted by illustrators for a long time [1]. In Figure 1 we can see an example of illustrative transparency as applied in a hand-crafted illustration. In this drawing, the opacity of the outer object is high near the silhouette and decreases smoothly toward the inner part of the object in order to create a strong depth ordering cue.

The goal of illustrative visualization is not to render images in a physically correct way, but to convey information. On the other hand, the HVS is accustomed to perceive physically correct images most of the time. In this paper, we do not want to create purely non-photorealistic (NPR) renderings since this rendering style tends to remove much of the visible features. For example, we are not interested in approaches that reduce the shape to a set of important lines. We believe it is important to retain the information available from shading. Based on these conditions, we discuss relevant findings from perception research, where the importance of so-called X- and T-junctions for the perception of transparency is known for a long time. Recent results were even able to find neurons with specific response patterns to a display of squares that form a X-junctions in the image [2]. Therefore, the motivation for our approach is two-fold: it is inspired by hand-drawn illustrations and also motivated by positive results from perception research.

• *The authors are with Scientific Visualization Group in the Department of Computer Science at ETH Zürich, Switzerland and Universitätsklinikum Marburg, Germany.*
E-mail: {crobi, rafuchs, jangy, peikert}@inf.ethz.ch, stephanie.mehl@med.uni-marburg.de

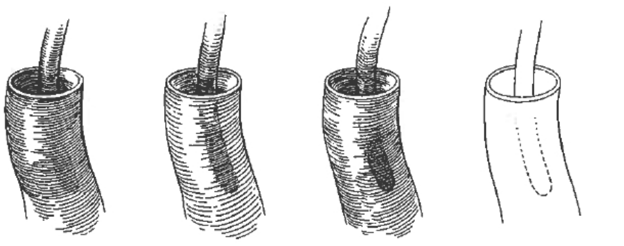


Fig. 1: Example of artistic drawings. The occluded object is hidden near the edge and its visibility increases with the distance to the edge. This effect is combined with four different shading styles. ©Gerald P. Hodge

2 Related Work

In this section we discuss a selection of the related work to provide context for the contributions presented in this paper.

Illustrative techniques: Illustrative techniques for the visualization of complex objects have become a popular research topic during the past decade. Similar to our method, Diepstraten et al. [3] discuss the application of transparency as suggested by Hodges [1]. However, their approach can only handle two transparent layers and is based on expensive object space distance computations. Luft et al. [4] present the unsharp masking of the depth buffer to add depth cues to images. Unlike our approach, their method modifies the color of the image and depends on the magnitude of the depth discontinuity at object silhouettes. Moreover, it is unclear how their method can be applied to transparent surfaces with multiple layers. Bruckner and Gröller [5] use volumetric halos to achieve a similar effect for volume rendering. In recent work Hummel et al. [6] present an illustrative approach for flow visualization based on user-placed cutting planes and surface streamlines.

Transparency: Transparent rendering is a common approach to visualizing occluded scene parts. However, a naïve use of transparency often results in images that are difficult to understand. Born et al. [7] suggest the use of normal variation to assign transparency for illustrative visualization of integral surfaces. Interrante et al. suggest to support the understanding of transparent surfaces by adding ridge and valley lines [8] and curvature-directed strokes [9]. Similarly, Lum et al. [10] suggest to augment transparent surfaces with animated particles to convey their shape. Weigle and Taylor [11] suggest glyphs and coloring to improve the understanding of multiple intersecting surfaces. In a seminal paper Chan et al. [12] present a method to minimize the discrepancy between actual and perceived transparency in volume renderings. The suggested methods rely on an expensive optimization scheme which requires several seconds per frame, whereas our method is interactive. Luboschik et al. [13] suggest to replace transparency by color weaving for 2D shapes and present a user study which evaluates shape recognition and interactive picking with respect to task-completion timings and errors. In recent work, Busking et al. [14] present an interesting algorithm based on depth-peeling for the image-based rendering of intersecting surfaces.

Lines: Line-based techniques are important because they use pixels very economically without occluding inner structures. Many illustrative techniques add lines to highlight the structure of the rendered objects. Saito and Takahashi [15] suggest a screen-space rendering approach to highlight discontinuities, edges, and contour lines, where data about the geometric properties are represented as a G-buffer. Nienhaus and Döllner [16] present blueprints, a technique for enhancing occluded features using edge maps. Other popular methods for detecting important feature lines include apparent ridges [17], Laplacian lines [18], and suggestive contours [19]. Complex objects often produce a dense set of feature lines which is difficult to understand. Appel et al. [20] propose the haloed line effect to improve the perception of the relative depth ordering of lines. Everts et al. [21] improve the effect by using depth-dependent halos. Similar to our method, Hamel et al. [22] use the priciples of Hodges [1] to enhance line drawings.

Shading, coloring, and texturing: In illustrative visualization, several approaches have been proposed to improve the perception of surface shape and details. Since our method does not change the actual surface color or lighting, they could be combined with our method. Gooch et al. [23] present a shading model for technical illustrations that uses both luminance and changes in hue to indicate surface orientation, reserving extreme lights and darks for edge lines and highlights. Vergne et al. [24] suggest a modified lighting model to enhance surface details, which uses a diffusion scheme to obtain a consistent global effect. Wong et al. [25] present texturing and coloring approaches to display layered information. Taylor [26] proposes a technique to optimize color and texture to visualize multiple fields on the same surface. Wang et al. [27] present a novel color design approach for illustrative visualization.

GPU and graphics buffers: Due to its easy hardware implementation, many approaches for order-independent transparency rendering are based on the depth peeling technique by Mammen [28]. A disadvantage of this technique is that it requires multiple rendering passes. A different approach is followed by the A-buffer, introduced by Carpenter [29]. In this technique, the color buffer is replaced by a linked list of fragments for each pixel. Early GPU implementations of the A-buffer are presented by Bavoil et al. [30], and Myers and Bavoil [31]. However, both implementations can only store a limited number of fragments in each list. Yang et al. [32] present an unbounded GPU implementation of the A-buffer by exploiting recent advances in graphics hardware. This work is a major inspiration for the implementation of our technique.

Silhouettes: As discussed above, artists often enhance silhouette lines of the illustrated surfaces. Since there is a significant variability in terminology, we give here our definition of a *silhouette*, as illustrated in Figure 2. Traditionally, the silhouette is defined as the set of points where the surface normal is perpendicular to the view direction (blue lines) [33]. However, this definition does not capture the surface boundary (red lines), which intuitively belongs to the silhouette as well. Therefore, we define a *silhouette*

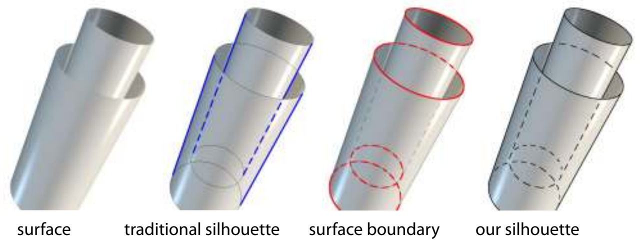


Fig. 2: Silhouette lines for two open concentric cylinders.

as the union of the two aforementioned sets (black lines).

Perception: Apart from being motivated by hand drawn illustrations, our method is also supported by the findings of perception research. In particular, Wilson and Keil [34] explain that the perception of transparency relies, as does visual perception in general, upon context to determine the most likely interpretation. Figure 3 illustrates the importance of X- and T-junctions, which are the single most important monocular cue for transparency [34]. Albert [35] shows that also the perception of lightness is based on the relation of contours forming X- or T-junctions and that the HVS does not compute layered decompositions of luminance. Nakayame et al. [36] explain that the HVS functions effectively by employing "crude tricks" or heuristics, rather than performing a detailed and/or exhaustive interpretation of the scene with all of the information available. They show that transparency is not coupled strongly to real-world chromatic constraints since combinations of luminance and color that would be unlikely to arise in real-world scenes still give rise to the perception of transparency. From this we are motivated to apply non-local changes to the rendering of transparent surfaces even though such a configuration is unlikely to occur in reality. The approach suggested in this paper is also supported by work of Beck [37], who explains that the visual system appears to incorporate an assumption of balanced transparency. In cases of unbalanced transparency the visual system is able to choose an interpretation that in some sense is the simplest. Anderson [38] discusses the importance of perceived contours to understand depth relations in transparent surfaces. He states evidence that the HVS employs rules that combine contrast magnitude and contour continuity to decide the depth ordering of surfaces. This insight serves as an additional motivation for the contour enhancement as illustrated in Figure 3. In other words, he conjectures that the HVS treats the highest contrast portion of a contour as a region in plain view. This relationship is exaggerated by our non-local transparency approach.

3 Overview

From the discussed results of perception research we can draw some conclusions: First, the heuristics applied by the HVS are robust even under non-realistic conditions as long as contrast relations are kept intact. Second, X-junctions (Figure 3(a)) are important cues for transparency. Third, T-junctions (Figure 3(b)) are good for understanding which surface is above which. The strength of the depth-ordering cue provided by transparent occlusion is directly proportional to the degree of contrast reduction. In the limit case the depth-ordering cue is maximal for fully opaque surfaces, as in the case of a T-junction. Therefore, we can infer from perception research a motivation to keep the T-junction property of layered surfaces. X-junctions, on the other hand, give rise to the perception of transparency, therefore we suggest to depict crossings as a fusion of both cues (Figure 3(c)).

Illustrators use similar rules as we can see in Figure 1. We can see that the illustrators modify the transparency to improve the perception of layers behind. Transparency is decreased where shading is important, for example for boundary enhancement. To further improve the perception of the silhouettes, we enhance these further by not only decreasing the transparency on the boundary, but also by increasing the transparency behind a boundary such that the background becomes visible. This results in a halo-like effect (Figure 3(d)). The final result is illustrated in Figure 3(e).

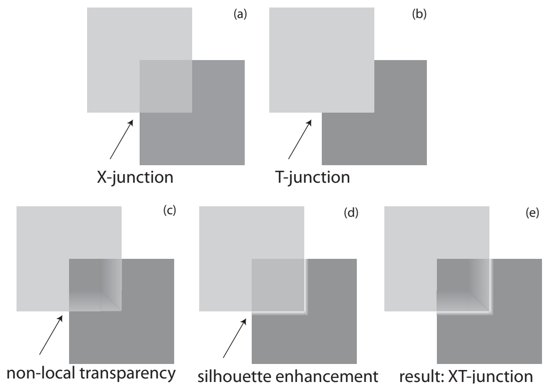


Fig. 3: The perception of transparency in the HVS is crucially dependent on the distinction between X- and T-junctions. (a) X-junctions evoke the perception of transparency. (b) T-junctions are best cues for depth-ordering. (c) Non-local enhancements to improve the perception of depth-layers. (d) Silhouette enhancement is important in the case of crossings. (e) We suggest a visual cue which combines properties of X- and T-junctions.

Based on the concept of XT-junctions, we design a novel *transparency enhancement method* for the rendering of layered surfaces. First, we define three transparency fields on the surface: one for the base transparency, one for the silhouette enhancement and one for the halos. The values for the two latter fields are fixed at the surface silhouette and a diffusion process is used to spread the information along the surface to the local neighborhood of the silhouettes. Finally, all three fields are combined and a modified alpha blending procedure is used to compute the final image color. The details of this method are described in Section 5.

The implementation of the transparency enhancement requires view-dependent and non-local information at each image point. In order to achieve interactive performance while maintaining full flexibility, we use a novel 2.5D image space representation of the whole scene. Similar to the A-buffer, our *illustration buffer* stores an unbounded linked list of all surface layers for each pixel. Unlike existing im-

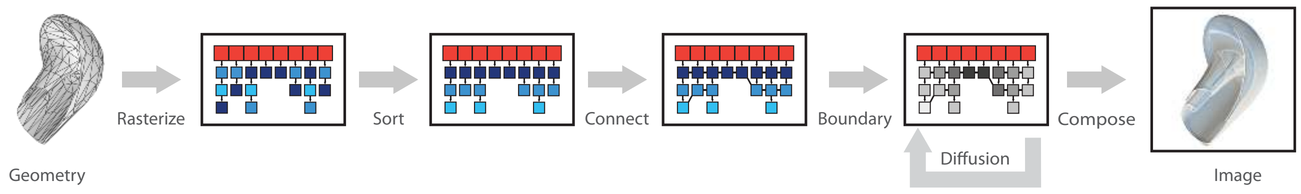


Fig. 4: Overview of our method. The input geometry is first rasterized into fragments which are then stored in our illustration buffer. This 2.5D image space representation is implemented on the GPU. All fragments are then sorted by depth and connected with geodesic neighbors. In the next step, we set boundary conditions and apply a transparency diffusion process based on results from perception research. Finally, all fragments in the illustration buffer are composed into the output image.

plementations, however, each fragment stores an arbitrary number of surface properties for deferred computations, as well as explicit links to geodesically neighboring fragments. Thanks to recent advances in graphics hardware, the illustration buffer is implemented on the GPU and can be used at interactive frame rates. Having simultaneous access to all fragments along a viewing ray as well as neighboring fragments along the surface makes our representation very flexible. It can not only be used to implement a wide range of existing transparency assignment techniques, but serves also as a base for novel 2.5D image filters. The illustration buffer is described in Section 4. Additionally, the work flow of our approach is illustrated in Figure 4.

Finally, in order to objectively assess the effectiveness of our method, we designed and conducted a rigorous *user study* measuring the understanding of complex surfaces from transparent renderings. In this study, three different tasks are used to quantitatively measure the task performance related to understanding of complex surfaces. The task scores are then used to compare our method to two other transparency assignment techniques. The study is described in Section 8.

4 Illustration Buffer

For the purpose of this paper, we define a pixel as the smallest addressable image element and a fragment as an intersection of the rendered surface with a ray going through the pixel center. Typically, surfaces are rendered as triangle meshes. Large triangles that cover multiple pixels in the image will therefore consists of multiple fragments. Conversely, small triangles that cover an area far smaller than a pixel might not generate any fragment at all. Some applications use multiple fragments for each pixel for the purpose of anti-aliasing (e.g., the supersampling technique). For the sake of simplicity, we do not discuss this case here, however, it would be straightforward to extend our method to handle multiple fragments per pixel.

Before describing the illustration buffer, we review the conventional way graphics cards generate images. Traditionally, graphics cards use two buffers to store the rendered image: the color buffer (or frame buffer) stores the color of each pixel and the z-buffer stores its distance to the viewer. Whenever an object is rendered, its polygons are projected onto the screen and rasterized into fragments. Each fragment then checks in an atomic way if it is closer to the viewer than the currently stored distance at the given pixel. If the fragment passes the z-buffer test, it updates both the color and depth information.

The illustration buffer uses a different approach, combining the ideas of the A-buffer and G-buffers. In this approach, each pixel stores a linked list of fragments. After the surface polygons are rasterized into fragments, each fragment is simply inserted into the respective list. In a final stage, each list is sorted by distance to the viewer and processed in order to determine the final pixel color. The following subsections describe in detail how the illustration buffer is built and used.

4.1 Buffer layout

The illustration buffer data structure consists of several separate buffers, which are all stored in graphics memory. Each buffer represents an array, whose elements store a fixed number of values. There are three types of buffers: those that contain a constant number of elements, those that contain one element per screen pixel, and those that contain one element per surface fragment. The following list describes the individual buffers. Their basic layouts are illustrated in Figure 5.

- `fragData` contains the data for all fragments. Each element in this buffer stores a number of application dependent properties of one fragment, such as the fragment position, normal direction, and color. This buffer is implemented as a 1D four channel texture object, with one buffer element spanning N consecutive texels, leaving room for $4N$ properties for each fragment.
- `fragData2` has the same layout as `fragData` and is used in ping-pong computation schemes.
- `fragNext` contains for each fragment the index of the next fragment belonging to the same pixel. This is similar to a “next” pointer in a linked list.
- `pixelHead` contains for each pixel the index of first fragment belonging to that pixel. This is similar to a “head” pointer in a linked list.
- `pixelCount` contains for each pixel the number of fragments belonging to that pixel.
- `fragCount` contains one single element, storing the total number of fragments written so far. This is also the index of the next free cell in the `fragData` array.
- `fragCountMax` is a constant number storing the size of the `fragData` buffer.

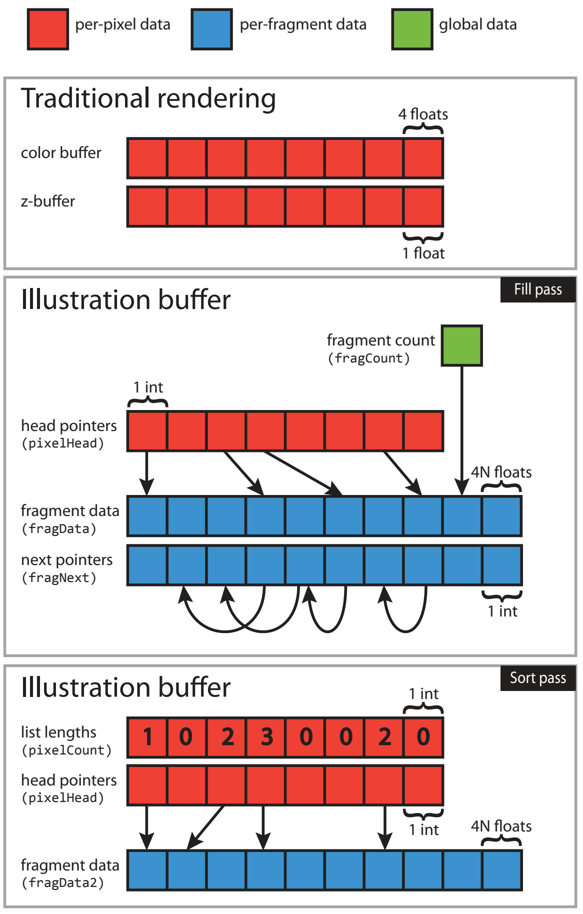


Fig. 5: The illustration buffer layout. Each red cell stores data of one pixel in the image, each blue cell stores data of one fragment. Green cells are global data.

4.2 Buffer filling

As in conventional rendering, polygons are first rasterized into fragments. This is done automatically by the graphics hardware. Each fragment is then simply inserted into the fragment list, using a fragment shader described in Algorithm 1. The shader consists of three important steps: First, `fragCount` is increased by one to get the index of a free cell in `fragData` (line 1). Next, the previous value of `pixelHead` is stored and replaced by the new index (line 3). After this step, `prevHeadId` and `newFragId` contain the old and the new fragment list heads, respectively. Finally, these two list elements are linked (line 4). The return value of this function is then the index of a free element that can be used to store all fragment properties.

The above algorithm needs several non-standard fragment shader operations, all of which are available in recent graphics hardware: atomic operations for protection against concurrent access, and read and write from arbitrary memory locations in a buffer. In our implementation, we access this functionality through OpenGL with the extension EXT_shader_image_load_store.

Note that `fragData` can only store a limited number of fragments and any additional fragments are simply ignored. After each rendering, we check if the number of generated fragments was larger than the buffer size. If this was the case, we resize the buffer to the required size and repeat the whole rendering process. This approach might be seen as wasteful of resources. However, we regard this limitation as temporary since current graphics hardware already contains specific constructs for dynamically sized buffers, available, e.g., in DirectX 11 as *append/consume buffers*.

Algorithm 1 Inserting a fragment into the illustration buffer

```
1: newFragId ← add(fragCount, 1)
2: if newFragId < fragMaxCount then
3:     prevHeadId ←
       exchange(pixelHead[x,y], newFragId)
4:     fragNext[newFragId] ← prevHeadId
5:     return newFragId
6: else
7:     return -1
8: end if
```

4.3 Fragment sorting

In the second step, the fragment lists are sorted by depth. This is implemented with a *full-screen render pass*, where a single quad covering the whole image is rendered. The fragment shader for this pass uses selection sort to construct the sorted sequence. This sort needs to repeatedly traverse the input fragment list. However, unlike previous methods based on insertion sort [32], it only uses one write operation per output list element and does not need a fixed-size local array. In our experiments, we have found no performance difference to the insertion sort method, presumably because repeated read operations benefit from caching.

At this point, the sorted elements could be used to compute the final pixel color. Unlike existing A-buffer implementations, however, we use the sorted lists in multiple subsequent passes and therefore we copy them back to global memory. For efficient access, the list elements are stored in consecutive buffer elements. Since this reordering cannot be done in-place, the second buffer `fragData2` is used to store the sorted lists. Similar to Algorithm 1, the corresponding shader program first reserves a block of free buffer elements. The index of the first element as well as the number of elements in the block are stored in `pixelHead` and `pixelCount`, respectively.

4.4 Neighbor search

In the third step, we connect each fragment to its four geodesic neighbors. More precisely, for a fragment with pixel coordinates (x,y), we search and store the index of the four nearby fragments with pixel coordinates $(x+dx, y+dy)$ with $(dx,dy) \in \{(\ 1,0),(1,0),(0,\ 1),(0,1)\}$ in the `fragData` buffer. This information is used later to access the surface neighborhood of a fragment. Since fragments at the boundary or silhouette of a surface have fewer than four neighbors, a special value is used to indicate missing neighbors.

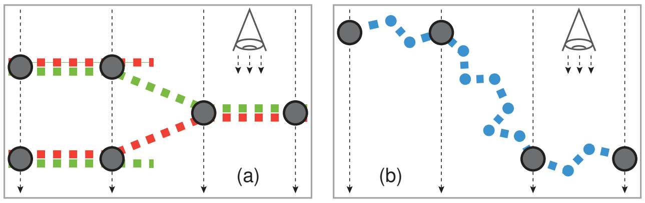


Fig. 6: Four neighboring viewing rays intersect the rendered surface at fragments (black circles). (a) Given these fragments alone, one cannot decide which ones are neighbors along the surface. Two possible cross sections of the original surface are indicated in red and green. (b) For finely triangulated objects and surfaces almost parallel to the viewing direction, two neighboring fragments might be separated by an arbitrary number of polygons.

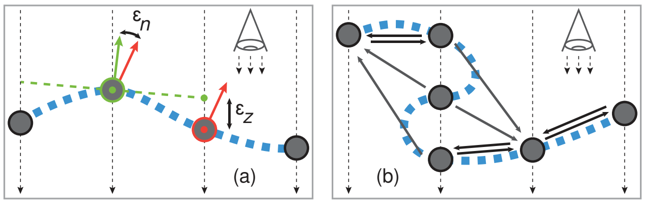


Fig. 7: (a) The difference between the fragment normal and eye distance gives a measure for how likely the fragments are neighbors along a continuous surface. (b) Arrows indicate the most likely neighbor according to the continuity measure. Fragments are marked as neighbors only if this relationship is mutual.

The problem of finding the correct neighbors with screen space information only is inherently difficult: in the situation depicted in Figure 6(a), one cannot decide how the fragments are connected. One difficulty is that during the rasterization of the triangle mesh into fragments, all connectivity information is lost. Additionally, using object space information is not simple: for finely triangulated surfaces, or for surfaces which are almost parallel to the viewing direction, neighboring fragments may lie on triangles which are very far apart (see Figure 6(b)). Since we cannot afford a global object space search for each fragment, we use a heuristic screen space method to decide which fragments are connected.

In practice, the visualized surfaces are often locally smooth and continuous. We therefore use a heuristic measure ε for the continuity of a fragment pair, as shown in Figure 7(a). This measure will be small for neighboring fragments along a continuous surface, and large for two fragments belonging to different surface layers. Given such a measure, a necessary condition for two fragments i and j being geodesic neighbors is that $\varepsilon(i,j)$ is smaller than the measure between i or j and any of the other neighbor candidates, i.e., that both fragments think of each other as the best candidates among the fragment list of the adjacent pixel. (see Figure 7(b)).

Our neighbor search is based upon the above observations and is again implemented in a full-screen render pass. The fragment shader traverses the fragment list of the current pixel and for each fragment, uses Algorithm 2 to find the corresponding geodesic neighbor. This search is repeated for each adjacent pixel. The continuity measure

Algorithm 2 Finding geodesic neighbors

```
1: fragNeighbor ← -1 {current neighbor candidate}
2: eNeighbor ← ∞ {measure $\varepsilon$ for fragNeighbor}
3: {find the best candidate for A among all B's}
4: for all fragB ∈ fragListB do
5:     eB ← $\varepsilon$(fragA, fragB)
6:     if eB < eNeighbor then
7:         eNeighbor ← eB
8:         fragNeighbor ← fragB
9:     end if
10: end for
11: {check if there is a better candidate among all A's}
12: for all fragA2 ∈ fragListA do
13:     eA ← $\varepsilon$(fragA2, fragNeighbor)
14:     if eA < eNeighbor then
15:         return -1
16:     end if
17: end for
18: return fragNeighbor
```

$\varepsilon(i,j)$ used in our neighbor search is given by

$$\varepsilon_z(i,j) = \frac{1}{r_{obj}}\left[z_i + (\mathbf{x}_j - \mathbf{x}_i)\cdot\left(\frac{dz}{d\mathbf{x}}\right)_i - z_j\right] \quad (1)$$

$$\varepsilon_n(i,j) = 1 - \mathbf{n}_i \cdot \mathbf{n}_j \quad (2)$$

$$\varepsilon(i,j) = w_z \cdot \varepsilon_z(i,j) + w_n \cdot \varepsilon_n(i,j) \quad (3)$$

Here r_{obj} is the radius of the bounding sphere of the rendered object and $\mathbf{x}_i$, $\mathbf{n}_i$, z_i, and $(\frac{dz}{d\mathbf{x}})_i$ the pixel coordinates, normal, eye space z coordinate, and z coordinate gradient of fragment i, respectively. Note that the gradient corresponds to the slope of the triangle that produces the fragment and can be computed efficiently with the GLSL shader instruction *dFdx*. In our measure, ε_z is a second-order difference in the normalized fragment z coordinate, and ε_n a function of the angle between the fragment normals. Both components are small for densely sampled neighboring fragments on a smooth surface. However, due to noise and discretization errors, none of them is reliable on its own. In our applications, we have achieved best results with a weighted average error with $w_z = w_n$. For objects with multiple connected components, we additionally store the component ID for each fragment and set ε to infinity if the two fragments belong to different components.

Note that our algorithm for the neighbor search can potentially introduce errors. The first type of error results from the loss of geometric information during the rasterization: if there is a sub-pixel size surface gap between two neighboring fragments, Algorithm 2 will still connect the fragments as if the surface was continuous. The second type of error is introduced by our heuristic measure. For very noisy surfaces, the measure could classify the wrong fragment pair as being closest along the surface. In this case, other measures can be used, such as the fragment distance in parameter space. This measure is particularly interesting for integral surfaces, which often have a natural global parametrization. In practice, we have found our neighbor search to be robust enough for the practical use, demonstrated by the absence of artifacts or flickering in animated scenes.

4.5 Operators

After the neighbor search, the illustration buffer contains a complete and general screen-space representation of the rendered surface, where each fragment has access to its geodesic neighbors along the surface as well as its neighboring fragments along the viewing ray. This representation can be processed by any number of purely local, local, or global operators, as discussed below. The choice of operators depends on the application and is discussed in Section 5. The following paragraphs therefore only give a definition of the three operator types.

Purely local operator: A purely local operator is a function that takes the fragment list of a pixel and computes new values for some properties for each fragment in that list. For example, a purely local operator could implement angle-based transparency [6] by setting the fragment transparency to the angle between surface normal and viewing direction. See Figure 11 for an example. Such an operator can again be implemented in a full-screen render pass with an appropriate fragment shader.

Local operator: A local operator is similar to a purely local operator, except that it uses information from neighboring fragments, as described in Section 4.4. Since updating the fragment properties could affect the input of other concurrently processed fragments in an unpredictable way, the two fragment data buffer `fragData` and `fragData2` are used in a ping-pong computation scheme. Examples of local operators include image filters such as the Sobel operator. Local operators can also be used for iterative implementations of global operations, such as image diffusion, and can therefore be repeated for a user-specified number of iterations.

Global operator: A global operator is a function that takes as input all fragment lists in the illustration buffer. Similar to a local operator, two fragment data buffers have to be used in a ping-pong computation scheme. Since memory bandwidth and latency are the two major bottlenecks in our method, care must be taken to limit the actual amount of input data consumed by such an operator. We do not use global operators in our visualization, however, they could be useful in other methods such as a multilayered screen space global illumination.

4.6 Fragment compositing

The last step in our method takes the fragment list of each pixel and computes the final pixel color. The details of this operation are application dependent. For example, opaque rendering sets the pixel color to the color of the first fragment in the list. Another very common example is alpha blending, where the final color is a linear combination of all fragment colors. Similar to a local operator, this step is implemented in a full-screen render pass with an appropriate fragment shader. Note that having simultaneous access to all surface layers in a single shader pass enables compositing operations that can not be easily implemented using conventional rendering and depth peeling. Additionally, since depth peeling renders the whole geometry each time a surface layer is extracted, compositing operations that traverse the fragment list multiple times might result in a large rendering overhead.

5 Non-local transparency enhancement

The previous section described how to build the illustration buffer data structure. This section describes how we use this data structure to implement an illustrative transparency assignment method that improves the understanding of transparent surfaces. Based on the result of previous perception research, we have extracted a set of simple rules for the surface transparency:

1) The transparency of the surface is set to zero at the silhouette and increases slowly with the distance to the contour. This enhances all surface silhouettes.
2) The opacity of the surface is set to zero wherever it is occluded by another surface silhouette and increases rapidly with the distance. This will create narrow halos around occluding surfaces. The increased contrast further enhances the perception of T-junctions.
3) The transparency should be a smooth function of the surface, except at the points described above. A discontinuity could easily be mistaken for a surface contour.

The first two rules are a straightforward application of our concept of the XT-junction (see Figure 3). Note that the first rule also corresponds to the rules postulated by Hodges (see Figure 1). The third rule was chosen to prevent high frequency changes in the transparency to be mistaken for shading details or surface shape.

These rules can be implemented with our illustration buffer approach. We use a purely local operator to set initial and boundary conditions for the transparency of individual fragments and then use a local operator to apply transparency diffusion. Similar to the solution of the heat equation after a finite time, we use the transparency diffusion with a fixed number of iterations to spread the boundary enhancement to the neighborhood of each silhouette.

5.1 Transparency fields

Because of the non-linear behavior of the transparency near junctions, we define three scalar fields on the surface:

- The initial transparency of the surface α
- A silhouette highlight field β, used to implement rule 1. This field will have high values at surface silhouettes and fall off with the distance to the silhouette.
- A halo highlight field γ, used to implement rule 2. This field will have high values near occluding edges and fall off with the distance to those edges.

In order to identify boundary fragments for each of the above fields, we also store binary fields b_α, b_β and b_γ with values of 0 for boundary fragments and 1 for all other fragments.

The values of the three fields α, β, and γ are stored as custom fragment properties (see Section 4.1). Since the scalar fields only take values from zero to one and b is a binary value, we can store both values in one scalar with minimal precision loss, e.g., as $2 \cdot b_\alpha + \alpha$.

5.2 Initial conditions

All fragments are first initialized with $\beta = 0$, $\gamma = 0$, $b_\alpha = 1$, $b_\beta = 1$, and $b_\gamma = 1$. For the initial value of α, we let the user choose among two different styles, defined by one of the following equations, where i is the index of the fragment in the sorted fragment list, n the number of fragments at the given pixel, and $s \in [0,1]$ a user-specified parameter.

$$\alpha_i = s \tag{4}$$

$$\alpha_i = 1 - (1-s)^{\frac{1}{n}} \tag{5}$$

Equation 4 assigns a constant initial transparency to each fragment and therefore produces results most similar to traditional alpha blending. A disadvantage of this approach is that regions with many layers will appear too opaque, while regions with only a few layers will appear washed out. Therefore, we additionally provide an adaptive transparency defined by Equation 5. This approach initializes the transparency so that the accumulated color intensity in the final image remains constant regardless of the number of layers, i.e., it solves the following problem:

$$\sum_{i=1}^{n} (1-\alpha)^{i-1}\alpha = s \tag{6}$$

5.3 Boundary conditions

After the initial transparency values are set, a local operator classifies the fragments to find fragments at the boundary of the surface. Since this operator does not depend on the output of the previous step, it can be implemented in the same rendering pass as the operator that sets the initial conditions. The boundary conditions are set according to the following rules, illustrated in Figure 8:

- If the fragment has fewer than four neighbors, it is a boundary fragment (red cell). Set $\beta = 1$ and $b_\beta = 0$.
- If the fragment layer index is smaller than the index of one of its neighbors, it is adjacent to a silhouette fragment (blue cell). Set $\gamma = 1$ and $b_\gamma = 0$.
- If the fragment layer index is greater than the index of one of its neighbors, it lies directly underneath some silhouette fragment (green cell). Set $\beta = 0$ and $b_\beta = 0$.
- Otherwise, it is an ordinary fragment (gray cells). No changes to the initial values are made in this case.

These boundary conditions are required for the subsequent diffusion step.

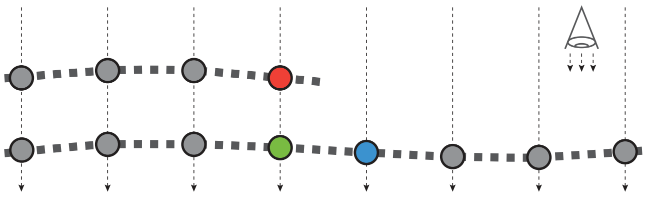


Fig. 8: Cross-section of the rendered surface. Boundary conditions for the transparency are set at the red, green and blue fragments.

5.4 Transparency diffusion

In order to smooth the initial transparency α according to rule 3, we apply a homogeneous diffusion defined by

$$\frac{\partial}{\partial t}\alpha = \lambda_\alpha \Delta\alpha \tag{7}$$

with a diffusion coefficient λ_α. We do not reach the steady state of the diffusion process, but instead stop after a given time. The diffusion process is implemented as a local operator using a forward discretization of Equation 7:

$$\alpha^{k+1} = \alpha^k + b_\alpha \lambda_\alpha \Delta\alpha^k \tag{8}$$

with a 2nd order central finite difference approximation of $\Delta\alpha$. In our default configuration, we use 10 to 50 iterations with a diffusion coefficient of $\lambda_\alpha = 1$.

The same diffusion process could be used to spread the initial values of β and γ to the neighborhood of silhouettes and occluding edges. However, the homogeneous diffusion transports values very inefficiently. Instead, we use a non-physical process

$$u^{k+1} = \max\{u^k, \max\{u^k_{neighbor}\} - \lambda_u\} \tag{9}$$

where $u \in \{\beta, \gamma\}$ and $u_{neighbor}$ are the values of u for the neighboring fragments. Using this procedure, the values of $1-u$ will be approximately proportional to the distance to the nearest boundary.

After the diffusion process, transparency fields β and γ will have a characteristic profile with high values near a boundary and values falling off with the distance to the boundary. For artistic reasons, we apply one of the following functions to the transparency fields in order to modify this profile:

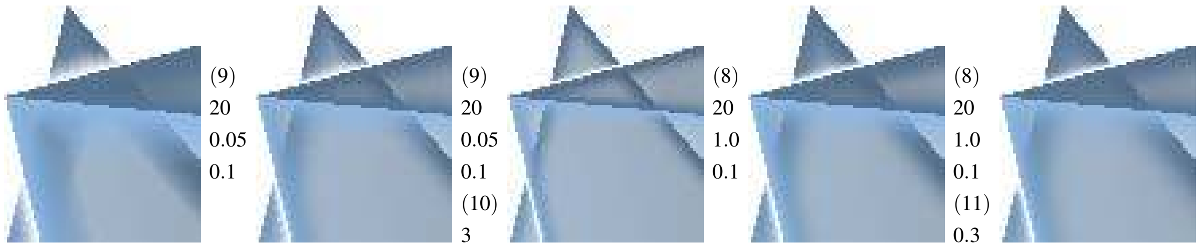
$$u \leftarrow \frac{1}{1-e^{-p}}\left[e^{-p(1-u)^2} - e^{-p}\right] \tag{10}$$

$$u \leftarrow u^p \tag{11}$$

Note that both functions are bijective on the range $[0,1]$. Figure 9 shows results of different configurations of parameters described in this section.

5.5 Fragment compositing

The three transparency fields α, β, and γ are combined in the final stage using a modified front-to-back alpha blending procedure. In this procedure, the transparency of a fragment is first initialized with α. The edge highlight field β is then multiplied with the remaining transparency $1-\alpha$ and added to the base value. Finally, the transparency



Diff.	(9)	(9)	(8)	(8)	(8)
$n =$	20	20	20	20	100
$\lambda_\beta =$	0.05	0.05	1.0	1.0	1.0
$\lambda_\gamma =$	0.1	0.1	0.1	0.1	0.1
Mod.		(10)		(11)	
$p =$		3		0.3	

Fig. 9: Different parameter configurations for the transparency diffusion. *Diff.* is the equation used for the diffusion process, *Mod.* is the equation used for modifying the field (if any), n is the number of iterations, and λ_β, λ_γ and p are the parameters for the diffusion, as described in Section 5.4

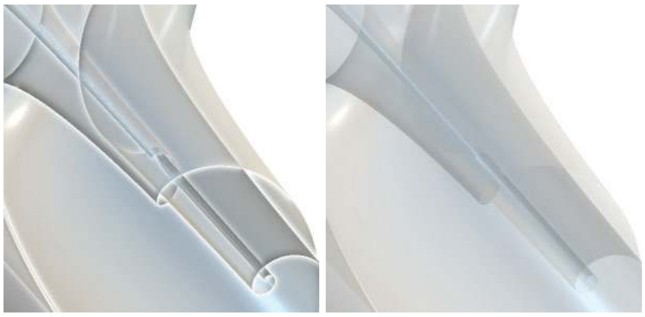


Fig. 10: A close up comparison of our method (left) with constant transparency (right).

Algorithm 3 Calculate the final pixel color

```
1: $\alpha \leftarrow 1$
2: $\mathbf{c} \leftarrow 0$
3: for $i \in [1..n]$ do
4:     $\hat{\alpha}_i \leftarrow (1 \quad \gamma_i)(\alpha_i + (1 \quad \alpha_i)\beta_i)$
5:     $\mathbf{c} \leftarrow \mathbf{c} + \alpha\hat{\alpha}_i\mathbf{c}_i$
6:     $\alpha \leftarrow \alpha(1 \quad \alpha_i)$
7: end for
8: $\mathbf{c} \leftarrow \mathbf{c} + \alpha\mathbf{c}_{background}$
9: return c
```

is multiplied with $1 \quad \gamma$ in order to account for halos. Note that since all input fields are smooth and we do not produce values outside of the acceptable range $[0,1]$, the resulting fragment transparency will be smooth along the surface.

Algorithm 3 illustrates our blending procedure, where n is the number of fragments for this pixel, $\mathbf{c}$ is the final pixel color, $\mathbf{c}_i$ is the color of i-th fragment and α_i, β_i, and γ_i are the values of the three fields as defined above. A close up comparison with constant transparency alpha blending is given in Figure 10.

6 Results

In this section we present images generated with our technique and compare our method with previous work.

Figure 11 shows a simple scene rendered with several common transparency assignment techniques [6]. While existing techniques only use local surface properties and are therefore very fast, they show various disadvantages. Opaque rendering (Figure 11(a)) has strong depth ordering cues, however does not show occluded objects. Constant transparency (Figure 11(b)) shows all occluded parts, but lacks depth ordering cues. Additionally, the transparency has to be made very high if one is to see all surface layers, which leads to a low contrast in the resulting image. Angle based transparency (Figure 11(c)) nicely highlights the silhouettes of curved objects, but not those of the flat cube. Similarly, normal variation transparency (Figure 11(d)) completely fails for the flat cube. Moreover, it is sensitive to noise and irregular tessellation due to the use of derivatives of the surface normal. Finally, our method (Figure 11(e)) properly highlights all silhouettes and maintains a high contrast while giving view of all surface layers.

Figures 12 and 16 show several examples from flow visualization. Such visualizations often contain complex surfaces for which one has to rely on the expressiveness of the rendering as it is difficult to apply context knowledge. Figure 12 shows a stream surface as well as an isosurface of vorticity in the simulation of two colliding vortex rings. Figures 16(a) and 16(c) show two different stream surfaces of a turbulent flow rendered with our method. Renderings of the same surfaces using conventional transparency assignment methods can be found in the supplementary material.

Figure 13 shows the visualization of a jet engine. Similar to blueprint rendering [16], Figure 13(a) was rendered with low constant transparency and enhanced with silhouette and crease lines. Figure 13(b) was rendered with normal variation transparency enhanced with haloed lines for increased depth ordering cues. Finally, Figure 13(c) was rendered with our method. Since the density of silhouettes is very high for this model, halos have been omitted to reduce visual clutter. Instead the same feature lines as in Figure 13(a) have been used. Even though all three methods show the overall structure of the engine and haloed lines provide strong depth ordering queues, our method additionally features unique stylistic elements such as a more uniform image brightness and a more visible surface shading around silhouettes due to increased opacity. All feature lines in these figures have been computed in object space and rendered as alpha blended triangle strips.

7 Evaluation

We have implemented our method as a custom rendering subsystem in the Visdom visualization toolkit [39]. Our implementation uses OpenGL 4.0 and its extension *GL_EXT_shader_image_load_store* for the concurrent global memory access as required for the illustra-

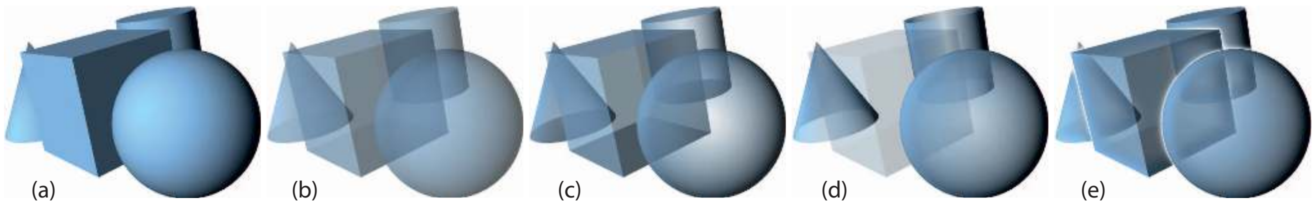


Fig. 11: A comparison of different transparency assignment methods: (a) opaque rendering, (b) constant transparency, (c) angle based transparency, (d) normal variation transparency, and (e) our method.

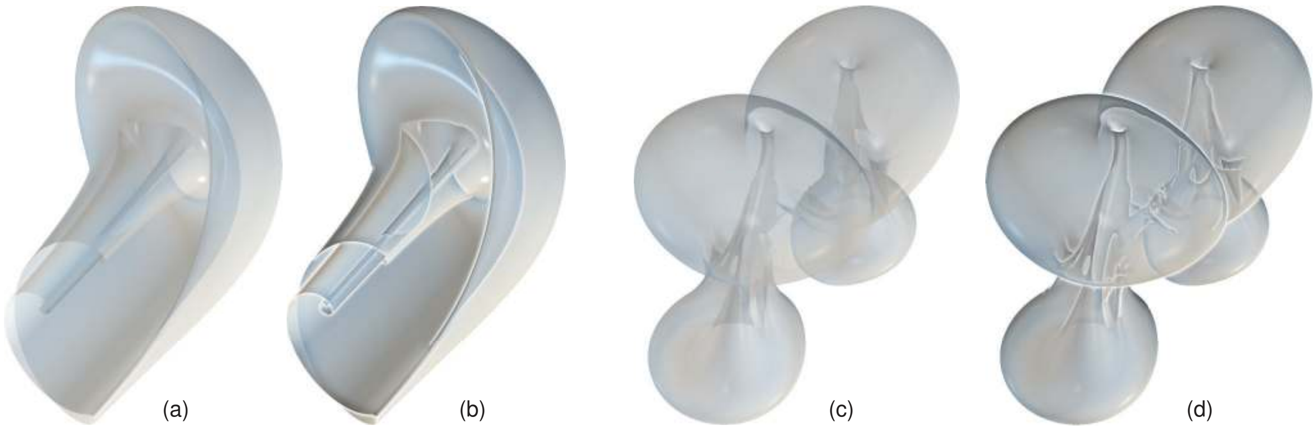


Fig. 12: A stream surface and an isosurface of two different turbulent flows. (a,c) Constant transparency. (b,d) Our method. Note the improved depth ordering cues at object silhouettes as well as an adaptively increasing transparency at points with many depth layers.

tion buffer. This extension is available on all consumer level hardware starting with the NVIDIA GeForce 400 and AMD Radeon 5000 series. Alternatively, OpenGL 4.2 core extensions *GL_ARB_shader_image_load_store* and *GL_ARB_shader_atomic_counters* could be used.

Figures 14 and 15 show the performance of our method in various settings. All tests were performed on a desktop computer with a NVIDIA GeForce 470 graphics card. The rendering time is broken down to the individual rendering passes *fill* (Section 4.2), *sort* (4.3), *connect* (4.4, 5.2, and 5.3), *solve* (5.4) and *compose* (5.5). The *clear* rendering pass is used to clear the contents of the illustration buffer. Each line represents the average time it takes to finish the respective rendering pass, including all previous passes. The measurements were performed by taking averages among 1000 images with a random camera position, so that the bounding sphere of the rendered object fills the entire image. Note that the *solve* pass corresponds to one iteration of our transparency diffusion. In practice, this pass will be repeated for a user specified number of iterations (typically 10 to 50), depending on the artistic preference and the resolution of the image. A comparison of different settings is shown in Figure 9.

Figure 14 presents the rendering time of the toroid surface as a function of the total number of fragments in the illustration buffer. The corresponding image resolutions range from 128×128 to 2048×2048. As expected from a screen-space method, our implementation scales linearly with the number of fragments in the scene. Note that for a square image, the number of fragments scales quadratically with the image width. This makes our method highly output sensitive: a 1024×1024 image with 50 iterations will take 20 times longer to compute than a 512×512 image with 10 iterations. In practice, applications should adapt the image resolution and/or the number of iterations to the desired level of interactivity. As an example, the scene from Figure 12(b) with a resolution of 512×512 pixels and 20 iterations runs at around 15 frames per second.

Figure 15 presents the rendering time of the toroid surface as a function of the total number of triangles in the surface mesh. The image resolution is set constant to 1024×1024 in this case. As seen from the timings of the *fill* pass, its run time has a constant component (indicated by the non-zero intercept) and increases approximately linearly with the number of triangles in the mesh. All other passes are independent of the input geometry.

For most images, the lengths of the fragment lists will vary across the image. The distribution of the list lengths depends on the shape of the input geometry and obviously influences the performance. However, we do not try an extensive evaluation of this effect since the distribution is hard to control without changing other parameters at the same time. In our examples, we have found the maximal list length (the number of passes required for depth peeling) to be usually several times higher than the average list length. In Figure 13(c), the average list length across the image is 5.64 while the maximal length is 95 (34 layers of geometry and 61 layers of feature lines). A similar variation was present in the toroid surface with a mesh resolution of 500K triangles. Even though most fragment lists contain only 1 or 2 elements, the average maximal list length (averaged over 1000 random camera settings) is surprisingly as high as 8.94. This can be explained by the fact that some isolated pixels depict parts of the surface which are almost parallel

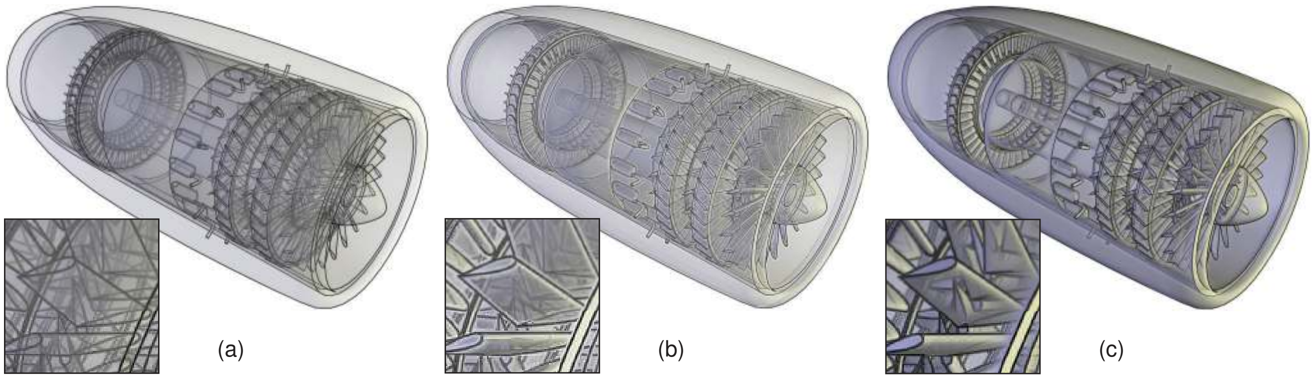


Fig. 13: Visualization of a jet engine model: (a) constant transparency, (b) normal variation transparency, and (c) our method. All methods have been enhanced with silhouette and crease lines.

to the viewing direction and contain many fragments of the very fine and slightly uneven surface mesh.

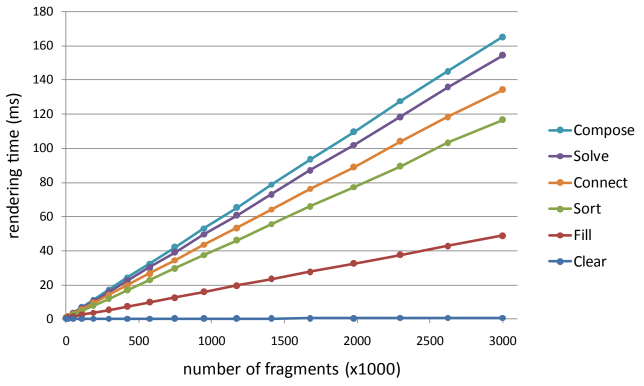


Fig. 14: Rendering time vs. number of fragments. Lines depict cumulative rendering times.

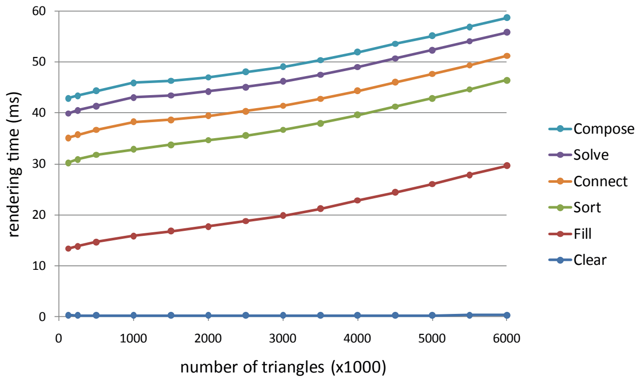


Fig. 15: Rendering time vs. number of triangles. Lines depict cumulative rendering times.

8 User Study

In this section we discuss the design and results of a user study performed to analyze the effectiveness of our illustrative transparency. The goal of this case study is to evaluate whether using the illustrative transparency presented in this paper instead of alpha blending or normal variation helps with the understanding of a three-dimensional flow surface.

8.1 Study Design

One important question in our study design is how to operationalize the concept of "understanding the visualization of a surface" so as to make it measurable [40].

Tasks: One problem is that other factors such as general intelligence or visual memory of the participants may constitute the main cause for variations in the test scores. Therefore, it is important to have multiple different tasks to balance such effects. In the present example we have selected the following tasks (see also Figure 16):

- Task 1 – layer index: The participant is asked to specify on which layer the surface silhouette at the given point is located. For example in Figure 16(a), the correct solution for A is two, since A is on the second layer.
- Task 2 – follow the boundary: The participant is asked to follow the shortest path on the boundary of the surface from a given start point to a given end point. The participant is asked to mark all other points encountered on the way. For example in Figure 16(b), the correct solution for the path from A to B is Y.
- Task 3 – nearest point: The participant is asked to specify which point is geodesically closest to another given point. For example in Figure 16(c), the nearest point to A is X.

These tasks have a precise definition, a unique solution, and can be solved within a short time frame. Of course, each of these tasks could be better solved by presenting a specialized visualization; however, this affects their value as an operationalization for the presented technique. Note there might be other tasks suitable for this user study. We do not claim here that our choice of tasks is optimal. For each task we present multiple sets of given points (chosen manually such that the task is sufficiently difficult) and calculate the score of a participant in this task as the number of correct answers.

Consideration of confounding factors: Another issue of a user study is the presence of confounding factors [41]. When multiple variables of the rendering such as transparency, texturing, coloring, and shading vary at the same time, it is impossible to understand the importance of each variable in separation. Therefore we modify only the

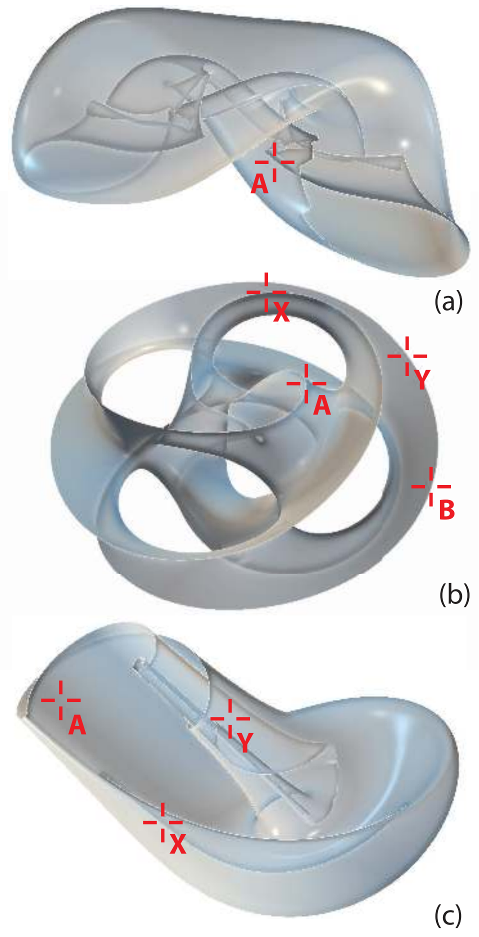


Fig. 16: Examples of the three tasks of the user study. (a) Task "layer index" with the "Pretzel" surface. (b) Task "follow the boundary" with the "Toroid" surface. (c) Task "nearest point" with the "Bubble" surface.

transparency of the surfaces and keep other factors such as viewing parameters, texturing, coloring, or shading fixed.

Study execution: To avoid surface shape training effects over the tasks, we use three different flow surfaces "Pretzel", "Bubble", and "Toroid", which are shown in Figure 16. This way each task can be performed on a different scene. During the user study, tasks are in a fixed order and each task is performed for one scene and one transparency assignment method. For example, one participant receives the first task on scene "Pretzel" rendered with alpha blending, the second task on the scene "Bubble" with normal variation, and the third task on the scene "Toroid" with illustrative transparency. The order of the factors "rendering method" and "scene" are counterbalanced [41] between the subjects. Renderings of the scenes based on alpha blending and normal variation can be found in the supplementary material. Since time consumed to understand a complex surface is an important factor, the time for solving each task was strictly limited to two minutes. Additionally, the number of points in each task was chosen so that it is realistically impossible to answer all of them within the given time limit. Missing answers were treated the same as wrong answers.

	Normal Variation	Alpha Blending
average	0.234	0.264
variance	0.035	0.051
degrees of freedom	68	
t-Statistics	0.62	
$P(T<t)$ one-sided	**0.270**	
	Alpha Blending	Illustrative Transparency
average	0.264	0.382
variance	0.051	0.077
degrees of freedom	67	
t-Statistics	-1.98	
$P(T<t)$ one-sided	**0.026**	
	Normal Variation	Illustrative Transparency
average	0.234	0.382
variance	0.035	0.077
degrees of freedom	61	
t-Statistics	-2.66	
$P(T<t)$ one-sided	**0.005**	

TABLE 1: Comparison of the percentage of correct answers for illustrative transparency, normal variation and alpha blending using paired samples t-tests.

8.2 Study Results

The total sample includes 36 students (4 female, 32 male) from the ETH Zürich, aged between 19 and 42. The most important question for the user study is the difference in the task performance that is attributable to the different transparency assignment methods. Our starting hypothesis is that illustrative transparency allows participants to perform better than normal variation and alpha transparency. In a first step, we check the data for outliers, but all scores are within the acceptable range.

The data was analyzed using paired samples t-tests comparing the percentage of correct answers. Results of paired samples t-tests [41] show that performance of subjects that were presented with alpha blending is comparable to the performance of subjects that were presented with normal variation ($p = 0.54$). In other words, the user study does not find a significant difference between alpha blending and normal variation. On the other hand, the difference between subjects that were presented with illustrative transparency compared to normal variation is highly significant ($p = 0.005$). The performance difference between illustrative transparency and alpha blending ($p = 0.026$) is significant as well. Table 1 gives an overview of these results, together with the average percentage of correct answers and its variation. We repeated the analysis of group differences using non-parametrical tests and obtained comparable results.

9 Conclusion

In this paper we have presented an illustrative transparency assignment method which is motivated from two directions. Perception research suggests that X- and T-junctions are the most important clue for the understanding of transparent surfaces. Therefore we suggest a hybrid cue which we call XT-junction. This approach is also motivated by a traditional drawing method of scientific illustrators, who also use halos and non-local transparency modulation in their illustrations. In a user study we have shown that this approach significantly improves the understanding of surfaces.

Since our method uses a non-local enhancement of surface silhouettes, it is not suitable for highly fragmented objects or surfaces with a dense set of silhouettes. In the future, we wish to address this issue by applying an anisotropic diffusion model that takes the local feature size into account. Another limitation is the use of a heuristic error measure to find geodesically neighboring fragments. Although we found it to be robust, it is possible to think of cases where it would not work. A surface parametrization could be used in this case to compute the geodesic distance between fragments in parameter space. We employ an iterative method to obtain the non-local distribution of transparency. While this method is quite simple to implement, faster convergence could be achieved with a more sophisticated multi-scale method which takes the geometry topology and texture coordinates into account.

This paper focuses on transparency while keeping other important cues such as texture, lighting or color fixed in order to allow a meaningful evaluation of transparency. We believe a decoupling of these parameters is necessary to develop a novel framework for transparency in this paper. On the other hand, this also forces us to limit the conclusions to the importance of transparency alone; we cannot make any speculations on the relative importance in relation to other properties such as surface texture for example. In future work we plan to extend our approach to a framework which takes multiple cues into account and to perform a user study which can provide insight into the relative importance of the different cues.

Even though the user study used a small number of difficult tasks, we have obtained significant results. This was possible since we expected a large difference between the methods, especially for surfaces that are complex and difficult to understand. Using such surfaces is also appropriate since we suggest our method specifically for the visualization of complex surfaces. In future work we would like to address the issues of task difficulty and study a wider range of tasks to measure the effectiveness of illustrative visualization methods. Additionally, our paper-based study and the choice of tasks was very general, and we do not claim that it is an optimal operationalization of the problem of understanding complex surfaces. For practical evaluations, an interactive study with application-specific tasks might be more appropriate.

Our understanding of how to best visualize a complex surface is by no means complete yet, and we can expect important research in this direction in the future.

Acknowledgments

This work was funded by the ETHIIRA grant ETH-12 09-3 and grant No. 226042 (SemSeg) from the Future and Emerging Technologies (FET) programme of the European Commission.

References

[1] E. R. S. Hodges, *The Guild Handbook of Scientific Illustration*. Wiley, 2003.

[2] F. T. Qiu and R. von der Heydt, "Neural representation of transparent overlay," *Nature Neuroscience*, vol. 10, pp. 283–284, 2007.

[3] J. Diepstraten, D. Weiskopf, and T. Ertl, "Transparency in Interactive Technical Illustrations," *Computer Graphics Forum*, vol. 21, no. 3, pp. 317–325, 2002.

[4] T. Luft, C. Colditz, and O. Deussen, "Image enhancement by unsharp masking the depth buffer," *ACM Transactions on Graphics*, vol. 25, no. 3, pp. 1206–1213, 2006.

[5] S. Bruckner and E. Gröller, "Enhancing depth-perception with flexible volumetric halos," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1344–1351, 2007.

[6] M. Hummel, C. Garth, B. Hamann, H. Hagen, and K. I. Joy, "IRIS: illustrative rendering for integral surfaces." *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1319–28, 2010.

[7] S. Born, A. Wiebel, J. Friedrich, G. Scheuermann, and D. Bartz, "Illustrative stream surfaces." *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1329–38, 2010.

[8] V. Interrante, H. Fuchs, and S. Pizer, "Enhancing transparent skin surfaces with ridge and valley lines," in *Proceedings of IEEE Visualization 1995*, 1995, pp. 52–60.

[9] ——, "Illustrating transparent surfaces with curvature-directed strokes," in *Proceedings of IEEE Visualization 1996*, 1996, pp. 211–219.

[10] E. B. Lum, A. Stompel, and K.-L. Ma, "Using motion to illustrate static 3d shape–kinetic visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 9, no. 2, pp. 115–126, 2003.

[11] C. Weigle and R. M. Taylor, "Visualizing intersecting surfaces with nested-surface techniques," in *Proceedings of IEEE Visualization 2005*, 2005, pp. 503–510.

[12] M.-Y. Chan, Y. Wu, W.-H. Mak, W. Chen, and H. Qu, "Perception-based transparency optimization for direct volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, pp. 1283–1290, 2009.

[13] M. Luboschik, A. Radloff, and H. Schumann, "A new weaving technique for handling overlapping regions," in *Proceedings of the International Conference on Advanced Visual Interfaces*, 2010, pp. 25–32.

[14] S. Busking, C. Botha, L. Ferrarini, J. Milles, and F. Post, "Image-based rendering of intersecting surfaces for dynamic comparative visualization," *The Visual Computer*, vol. 27, no. 5, pp. 347–363, 2011.

[15] T. Saito and T. Takahashi, "Comprehensible rendering of 3-d shapes," in *Proceedings of the 17th annual Conference on Computer Graphics and Interactive Techniques*, 1990, pp. 197–206.

[16] M. Nienhaus and J. Döllner, "Blueprints: illustrating architecture and technical parts using hardware-accelerated non-photorealistic rendering," in *Proceedings of Graphics Interface 2004*, 2004, pp. 49–56.

[17] T. Judd, F. Durand, and E. Adelson, "Apparent ridges for line drawing," *ACM Transactions on Graphics*, vol. 26, no. 3, pp. 19–26, 2007.

[18] L. Zhang, Y. He, X. Xie, and W. Chen, "Laplacian lines for real-time shape illustration," in *Proceedings of the 2009 Symposium on Interactive 3D Graphics and Games*, 2009, pp. 129–136.

[19] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella, "Suggestive contours for conveying shape," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 848–855, 2003.

[20] A. Appel, F. J. Rohlf, and A. J. Stein, "The haloed line effect for hidden line elimination," *SIGGRAPH Computer Graphics*, vol. 13, no. 2, pp. 151–157, 1979.

[21] M. H. Everts, H. Bekker, J. B. T. M. Roerdink, and T. Isenberg, "Depth-dependent halos: Illustrative rendering of dense line data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1299–1306, 2009.

[22] J. Hamel, S. Schlechtweg, and T. Strothotte, "An approach to visualizing transparency in computer-generated line drawings," in *Information Visualization, 1998. Proceedings. 1998 IEEE Conference on*, 1998, pp. 151–156.

[23] A. Gooch, B. Gooch, P. Shirley, and E. Cohen, "A non-photorealistic lighting model for automatic technical illustration," in *Proceedings of the 25th annual Conference on Computer Graphics and Interactive Techniques*, 1998, pp. 447–452.

[24] R. Vergne, R. Pacanowski, P. Barla, X. Granier, and C. Schlick, "Light warping for enhanced surface depiction," *ACM Transactions on Graphics*, vol. 28, no. 3, pp. 25:1–25:8, 2009.
[25] P. C. Wong, H. Foote, D. L. Kao, R. Leung, and J. Thomas, "Multivariate visualization with data fusion," *Information Visualization*, vol. 1, no. 3/4, pp. 182–193, 2002.
[26] R. Taylor, "Visualizing multiple fields on the same surface," *IEEE Computer Graphics and Applications*, vol. 22(3), pp. 6–10, 2002.
[27] L. Wang, J. Giesen, K. T. McDonnell, P. Zolliker, and K. Mueller, "Color design for illustrative visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 14, no. 6, pp. 1739–1754, 2008.
[28] A. Mammen, "Transparency and antialiasing algorithms implemented with the virtual pixel maps technique," *IEEE Computer Graphics and Applications*, vol. 9, no. 4, pp. 43–55, 1989.
[29] L. Carpenter, "The a-buffer, an antialiased hidden surface method," in *Proceedings of the 11th annual conference on Computer Graphics and Interactive Techniques*, 1984, pp. 103–108.
[30] L. Bavoil, S. P. Callahan, A. Lefohn, J. L. D. Comba, and C. T. Silva, "Multi-fragment effects on the GPU using the k-buffer," in *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*. ACM New York, NY, USA, 2007, pp. 97–104.
[31] K. Myers and L. Bavoil, "Stencil routed A-Buffer," in *ACM SIGGRAPH 2007 sketches*, 2007.
[32] J. C. Yang, J. Hensley, H. Grün, and N. Thibieroz, "Real-Time Concurrent Linked List Construction on the GPU," *Computer Graphics Forum*, vol. 29, no. 4, pp. 1297–1304, 2010.
[33] T. Isenberg, B. Freudenberg, N. Halper, S. Schlechtweg, and T. Strothotte, "A developer's guide to silhouette algorithms for polygonal models," *IEEE Computer Graphics and Applications*, vol. 23, no. 4, pp. 28–37, 2003.
[34] R. A. Wilson and F. C. Keil, *MIT Encyclopedia of the Cognitive Sciences*. MIT Press, 2001.
[35] M. K. Albert, "Occlusion, transparency, and lightness," *Vision Research*, vol. 47, no. 24, pp. 3061–3069, 2007.
[36] K. Nakayame, S. Shimojo, and V. S. Ramachandran, "Transparency: relation to depth, subjective contours, luminance, and neon color spreading," *Perception*, vol. 19, no. 4, pp. 497–513, 1990.
[37] J. Beck, "Perception of transparency in man and machine," *Computer Vision, Graphics, and Image Processing*, vol. 31, no. 2, pp. 127–138, 1985.
[38] B. L. Anderson, "The role of occlusion in the perception of depth, lightness, and opacity," *Psychological Review*, vol. 110, no. 4, pp. 785–801, 2003.
[39] J. Waser, R. Fuchs, H. Ribicić, B. Schindler, G. Blöschl, and E. Gröller, "World lines," *IEEE Transactions on Visualization and Computer Graphics*, vol. 16, no. 6, pp. 1458–67, 2010.
[40] R. A. Bailey, *Design of Comparative Experiments*. Cambridge University Press, 2008.
[41] G. E. P. Box, J. S. Hunter, and W. G. Hunter, *Statistics for Experimenters: Design, Innovation, and Discovery*. Wiley-Interscience, 2005.

Robert Carnecky is a graduate student at the ETH Zürich, Switzerland. He received his masters in Computational Science and Engineering at the ETH in 2007. His research focuses on scientific and illustrative visualization.

Raphael Fuchs graduated in computer science (2006) at Philipps-University Marburg (Germany) under the supervision of Prof. Volkmar Welker and received his Ph.D. degree in computer science (2008) from the University of Technology Vienna (Austria) under the supervision of Prof. Helwig Hauser and Prof. Eduard Gröller. He is now with the Chair of Computational Science at the ETH Zurich (Switzerland). His fields of interest include scientific visualization and applications of machine learning to visualization.

Stephanie Mehl graduated in Psychology (2005) at Philipps-University Marburg (Germany) under the supervision of Prof. Dr. Cornelia Exner and Prof. Dr. Winfried Rief and received her Ph. D. in clinical psychology (2010) under the supervision of Prof. Dr. Tania Lincoln and Prof. Dr. Winfried Rief at the same university. She performed a qualification training as clinical psychologist at the department of Clinical Psychology, Philipps-University of Marburg and in the Department of Medicine, Hospital for Psychiatry and Psychotherapy at the Rheinische-Friedrich-Wilhelms-University, Bonn, Germany. She is currently working in the Brain Imaging Unit at the Department of Medicine, Philipps-University Marburg with Prof. Dr. Tilo Kircher. Her research interests are multivariate statistics, clinical psychology and social cognition in severe mental disorders (schizophrenia, dementia).

Yun Jang is an assistant professor of computer engineering at Sejong University, Seoul, South Korea. He received the masters and doctoral degree in electrical and computer engineering from Purdue University in 2002 and 2007, respectively, and received the bachelors degree in electrical engineering from Seoul National University, South Korea in 2000. He was a postdoctoral researcher at CSCS and ETH Zurich, Switzerland from 2007-2011. His research interests include interactive visualization, volume rendering, visual analytics, and data representations with functions. He is a member of the IEEE.

Ronald Peikert received his diploma in mathematics in 1979 and his PhD in mathematics in 1985, both from ETH Zurich. He is currently a titular professor in the computer science department of ETH Zurich. His research interests include flow visualization, feature extraction techniques, and industrial applications of visualization. He is a member of the IEEE Computer Society.