Illustrative Rendering of Transparent Surfaces

Author(s):
Waldin, Nicholas

Publication Date:
2013

Permanent Link:
https://doi.org/10.3929/ethz-a-009767842

Rights / License:
In Copyright - Non-Commercial Use Permitted
Illustrative Rendering of Transparent Surfaces

Nicholas Waldin
Supervisors: R. Carnecky, Prof. Dr. R. Peikert

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

ETHZ 2013
Abstract

Stream surfaces are used to visualize snapshots in time of fluid flow. However, as useful as they are, they also have significant problems. In particular, humans have a difficult time understanding these surfaces and often make errors.

Research indicates that the perception of a transparent surface involves more than just the actual physical properties of the object itself. In addition, the human mind isn’t simply seeing something, but is actually trying to construct what it believes to be seeing. This suggests using an approach to visualizing stream surfaces that is not directly based on physical properties.

This thesis investigates the hypothesis that artificial coloring can help viewers perceive stream surfaces. In this thesis I present three algorithms for deciding on and placing colors based on a surfaces curvature or the use of simulated viewpoints. I designed and carried out a user study to test how users perceive images of surfaces that were created with these algorithms. The results of the study, however, did not indicate that artificial coloring significantly increases the ability of viewers to perceive stream surfaces. While the study was not conclusive, it may be possible to design a better study that does show this. On the other hand, the study can be seen as support for another conclusion, namely, that color does not improve the perception of transparent surfaces.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>i</td>
</tr>
<tr>
<td>Contents</td>
<td>iv</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Related work</td>
<td>5</td>
</tr>
<tr>
<td>2.1 Transparency and Perception</td>
<td>5</td>
</tr>
<tr>
<td>2.2 Enhanced Silhouettes</td>
<td>5</td>
</tr>
<tr>
<td>2.3 Other Approaches</td>
<td>6</td>
</tr>
<tr>
<td>3 Description of the Algorithms</td>
<td>9</td>
</tr>
<tr>
<td>3.1 Overview</td>
<td>9</td>
</tr>
<tr>
<td>3.2 Color Space</td>
<td>9</td>
</tr>
<tr>
<td>3.3 Assigning Colors and Smoothness</td>
<td>9</td>
</tr>
<tr>
<td>3.3.1 computing $\beta$</td>
<td>10</td>
</tr>
<tr>
<td>3.4 Curvature Based</td>
<td>11</td>
</tr>
<tr>
<td>3.4.1 Calculation of Curvature</td>
<td>11</td>
</tr>
<tr>
<td>3.4.2 Placing Sources</td>
<td>12</td>
</tr>
<tr>
<td>3.4.3 Assigning Colors to Sources</td>
<td>14</td>
</tr>
<tr>
<td>3.4.4 Curvature Method Comparison</td>
<td>15</td>
</tr>
<tr>
<td>3.5 Grid Based</td>
<td>20</td>
</tr>
<tr>
<td>3.5.1 Mesh Analysis</td>
<td>20</td>
</tr>
<tr>
<td>3.5.2 Placing Color Sources</td>
<td>21</td>
</tr>
<tr>
<td>3.5.3 Grid Implementation Details</td>
<td>22</td>
</tr>
<tr>
<td>3.5.4 Grid Evaluation</td>
<td>25</td>
</tr>
<tr>
<td>4 Performance</td>
<td>27</td>
</tr>
<tr>
<td>4.1 System Specifications</td>
<td>27</td>
</tr>
<tr>
<td>4.2 Curvature Methods</td>
<td>27</td>
</tr>
<tr>
<td>4.3 Grid Method</td>
<td>27</td>
</tr>
<tr>
<td>5 User Study</td>
<td>29</td>
</tr>
<tr>
<td>5.1 Intent</td>
<td>29</td>
</tr>
<tr>
<td>5.2 Study Design</td>
<td>29</td>
</tr>
<tr>
<td>5.3 Task Explanation</td>
<td>29</td>
</tr>
<tr>
<td>5.4 Evaluation of Answers</td>
<td>30</td>
</tr>
<tr>
<td>5.5 Results</td>
<td>30</td>
</tr>
</tbody>
</table>
6 Conclusion
  6.1 Study ................................................. 33
  6.2 Possible Implications ............................... 33

7 Future Work ............................................. 37
  7.1 Curvature Method .................................... 37
      7.1.1 Distance Between Sources ..................... 37
      7.1.2 Sensitivity to Parameters ..................... 37
      7.1.3 Cutoff Value .................................. 37
  7.2 Grid Method ......................................... 37
      7.2.1 Multiple Rays ................................ 37
      7.2.2 Interactivity ................................ 38

8 Bibliography ............................................. 39

Appendix A .................................................. 41

Appendix B .................................................. 51

Statement ................................................... 57
1. Introduction

Stream lines and stream surfaces are used to visualize snapshots in time of fluid flow. Figure 1 from Born et al. is an example of such a surface created by the airflow over a delta wing. A stream line is tangent to the velocity vector field at one point in time. This means that \( ds \times v = 0 \), where \( ds \) is a length element along the streamline and \( v \) is the vector in the field. A stream surface is an extension of this. It can be viewed as an infinitely dense set of stream lines, and is created by seeding (along a line) and following numerous stream lines.

However, as useful as they are, stream lines and surfaces also have significant problems. There can be numerous vortices in these flows resulting in complex stream surfaces, as can be seen in the same delta wing image (Figure 1). Humans have a difficult time understanding these surfaces and often make errors. To put this in perspective, later in this thesis I present a user study that measured how well humans understand these surfaces. It is not uncommon for people to get less than 50% of the questions correct. Certainly, the percentage of correct answers depends heavily on the difficulty of the questions, but it is still a clear indicator that the viewers did not understand the entire surface correctly. Because they are hard to understand, it is unlikely that a viewer will understand all of a surface correctly. It is therefore essential that research is done in this area to improve the rendering methods of these surfaces; how can someone be expected to draw correct conclusions if his understanding of a surface is incorrect?

There have been other attempts to help viewers understand stream surfaces, such as adding lines to the surface and varying transparency based on normals, or enhancing silhouettes. Adding lines to a surface can lead to clutter and basing transparency on the normals of a surface can occlude important information. Other methods are based on cutouts or exploded views. But exploded views can lead to misunderstanding of the size of the surface and cutouts might remove important context information.

In my thesis I investigate adding color to stream surfaces in combination with the silhouette enhancement of Carnecky. I do not add color to indicate different velocities on the surface - such as red for high and blue for low velocity - but to aid a viewer’s perception of the surface. This means I intend to color the surface to optimally enhance a viewer’s understanding of it. To the best of my knowledge there has been no similar research using...
1. Introduction

Figure 2: All three images show several tubes (not stream surfaces). The top shows without any enhancement, the second with silhouette enhancement [3], the third with one of my coloring methods with silhouette enhancement.

this approach for aiding human perception.

Human perception of light and color can be notoriously fickle, for example two different surfaces can appear to have different shades of grey even if they have the same color [4]. This is not limited to shading; luminance can also trick the eyes on how transparent a surface appears [5]. In other words, a human may - and easily so - see something that is physically incorrect. This means it is possible for something to have a different effect than seems logical. In [4] it is indicated, although not demonstrated, that color may have little or no effect on transparent surfaces. Nevertheless it is possible that my hypothesis on coloring may work for one simple reason: if two layers have a different color, then the perceived difference is greater than without color, and a viewer is more likely to perceive them as separate.
Figure 3: All three images show a stream surface. One without enhancement, one with silhouette enhancement [3], and one combining one of my coloring methods with silhouette enhancement.
1. Introduction
2. Related work

2.1 Transparency and Perception

Metelli found the alpha transparency formula \(((1 - \alpha) \times t)\) in 1970 [paper], which forms the basis for modeling transparency. Since then a number of experiments have shown that this formula does not fully explain human perception of transparency. Mettelli himself points out that a black episcotister appears more transparent than a white one, even if they have the same physical transparency [6]. Anderson and Singh [5] showed that other factors such as luminance play a role in how transparent a surface appears to be; a surface seems more transparent when the area is more illuminated. Richards et al. [7] looked at how color is perceived when combined with transparency and have shown that the inferred color may not be physically realizable. In a survey in 2010, Kramer and Bressan [4] looked at how color is connected with transparency. They claim that the brain initialy perceives color and object seperately and then combines them to form an image. They cite some studies that indicate this, but do not present any new research to substantiate this.

The above research indicates that the perception of a transparent surface involves more than just the actual physical properties of the object itself. In addition, the human mind isn’t simply seeing something, but is actually trying to construct what it believes to be seeing. Neon spreading can be considered one such example [8]. This poses the questions: what pitfalls must we worry about, and how can we use human perception to help a person understand an object; rendering an object not according to its physical properties may lead to better results.

2.2 Enhanced Silhouettes

The approach taken in this thesis builds on Carnecky’s work [3] on choosing \(\alpha\) values for transparent surfaces and enhancing silhouettes. His method emphasizes silhouettes by changing the transparency of surface layers near them. The opacity of the silhouette’s surface is increased, while the opacity of other surfaces is decreased. This is achieved with the following three rules, which are a summary of those given by Carcnecky on page 7:

1. The surface has 0 transparency at the silhouette.

2. When a surface is occluded by a silhouette, the transparency of that surface is 1.

3. Transparency is smooth, except where rules 1 and 2 apply.

In an initial stage the \(\alpha\) of each surface point is set to \(1 - (1 - s)^{1/n}\) where \(s\) is user defined and \(n\) is the number of layers at that point. In the absence of other factors affecting the image (e.g. lighting), this means every pixel that the object occupies has the same color intensity. If this were not achieved, then some areas would appear to be washed out while others very dark. Then he applies rules 1 and 2 above. Afterwards, rule 3 is achieved.
by smoothing the transparency of the surface layers through diffusion and a non-physical process for the silhouettes. This increases the halo and thickness of the silhouettes, thereby increasing their visibility.

This results in images like these:

![Figure 4: Examples of the rendering method from Carnecky et al.][3]

### 2.3 Other Approaches

One way of making an image more understandable is to give a texture to surfaces, for example see [9]. These textures can take the form of stripes, dots, dashes, or anything similar. Born et al. applied this idea to the visualization of stream surfaces [1], by adding streamlines with arrows to the surfaces. This technique may lead to obfuscation and clutter, and a part of these papers deal with managing such clutter.
2.3 Other Approaches

Others have - more in line with some of the techniques used in this thesis - tried to alter the alpha transparency of different parts of an object or its silhouettes. For example Chan et al. [10] attempt to find optimized alpha values in volume rendered images. However the examples and experiments are of fish and other objects, not of stream surfaces. Another attempt was made by Hummel et al. [2] to combine textures on a stream surface with rendering techniques that emphasize the silhouettes. This has the same problem as mentioned, that is, it can lead to unwanted clutter. There is also the possibility that some contextual information is hidden.
2. Related work
3. Description of the Algorithms

3.1 Overview

There are two aspects to making stream surfaces more comprehensible. First, overlapping surfaces should have different colors. The colors should be distinct enough so that viewers can easily tell the difference. This also raises the question as to how many colors we should use. The second is to avoid abrupt changes in color, because these might be misinterpreted as the edge of a surface. In other words, the color must change smoothly. Any method must satisfy these conditions. In this thesis I develop three methods for coloring surfaces. The first two are based on curvature, and the third uses a grid technique that is similar to ray casting.

The remainder of this section is structured as follows. First I describe the color space and how I chose colors so that the colors on the surfaces are distinct. Then I describe the formula I use to make surface colors smooth. Both of these techniques are common to the three methods. This is followed by a section describing the two curvature methods and a section describing the grid method.

3.2 Color Space

There is a tradeoff between the number of colors used and the difference between them. The more colors are used, the less difference exists between them. In this thesis I used a vector space with the three primary colors as basis vectors, which are represented by [1,0,0],[0,1,0] and [0,0,1], for red, green, and blue, respectively. Any color is a linear combination of these three. However one can not just take the distance between two colors in this vector space to determine how distinct they are to humans. Furthermore just using the three primary colors is not sufficient. Through experimentation I chose to use 6 colors with the RGB values (0,0,1),(0.784,1,0),(1,0,0),(0.231,0.494,0),(1,1,0),(0.933,0.51,0.933). More than 6 colors leads to colors that are hard to distinguish.

3.3 Assigning Colors and Smoothness

The technique I use to provide smooth changes of color on the surface is to select certain points to be “sources”, assign one of the six colors described above to each of these, and distribute colors based on mix of these sources. How sources are selected depends on the method used and is described in the corresponding sections. The color of a point on the surface is a combination of the colors of the source points: basically, the further away a point is from a source, the less of its color it receives. Thus, choosing the color for a source depends on the method used, but determining the amount of color a point receives from a source does not. In the implementation, surfaces are represented as a mesh. Distinguished vertices in the mesh are chosen as sources, and all vertices receive a color. To calculate the amount of color any vertex receives, I define a function $f(x_i, c(x_i), v_j)$, where $x_i$ is the
vertex of color source i, \( c(x_i) \) is the color of that color source, and \( v_j \) is the vertex to which we are assigning the color. The function calculates the geodesic distance between \( x_i \) and \( v_j \) and assigns an amount of color \( c(x_i) \) to \( v_j \). The geodesic distance is normalized for each color source and the color of \( v_j \) equals \( \beta \times c(x_i) \). We color even the source vertices like this, because if we didn’t, it may be possible that a source vertex could have a vastly different color from its neighbors.

### 3.3.1 Computing \( \beta \)

\( \beta \) can be set to decrease fast or slow, linearly or exponentially. We can also define a minimum \( \beta \), i.e. after a certain distance the factor would be constant. The effects are fairly clear: if \( \beta \) decreases too quickly, then only the points in the near vicinity of the sources will have a color; the others will either be black or nearly black. On the other hand, if \( \beta \) decreases too slowly, then the result will be mostly a mix of the different colors, leading to a fairly monochromatic mesh. Another issue is how close the different color sources are to each other. We may have color sources that are very close to each other, while others might not have any neighbors for quite some distance: we need to look at neighboring color sources when we calculate \( \beta \). Lastly \( \beta \) must be smooth and continuous, otherwise there might be sudden jumps in color. I define \( \beta \) in the following manner:

\[
\beta = \gamma^{-\text{distance}} \\
\gamma = \text{minvalue}^{-1/\text{mindistance}}
\]

where \( \text{mindistance} \) is the distance between the source and its nearest neighboring source, and \( \text{minvalue} \) is defined by the user as the value \( \beta \) has when it reaches that distance. This allows the function to be aware of its local neighborhood while retaining smoothness. The user is allowed to set a minimum value if he desires, below which \( \beta \) may not drop. This can be useful in that no surface appears black.
3.4 Curvature Based

The curvature of a stream surface allows us to partially understand a local area of the surface: For example, if the curvature is high, then the stream surface is “turning”, e.g. we could be near a vortex. On the other hand, low curvature indicates a flat area. Since this ties in with the overall structure of the surface, it leads to two different areas where we can place color sources: low and high curvature areas. After all, if we want to put sources in relatively flat areas in order to separate them, we simply need to look for areas with low curvature. If we want to put them where there is a lot of twisting and turning, so that they are marked and stand out from the more flat areas, we look for areas with high curvature.

This of course raises the question of how we determine high and low curvature. The method I used in this thesis is described in Section 3.4.1. Then we must locate the vertices to be used as sources, which is described in Sections 3.4.2 and 3.4.2. Finally we have to assign colors to these sources. This is described in Section 3.4.3. At this point coloring the surface follows the description in Section 3.3. We conclude the discussion of curvature in Section 3.4.4 where we compare the different methods.

3.4.1 Calculation of Curvature

If a surface is described by an analytical function, there is only one way to calculate curvature. In contrast, there are several different methods of calculating curvature on a mesh. I chose to use the Gaussian curvature. There is no numerical reason why this was chosen over other methods; but it is simple to calculate and is commonly used. It is possible for Gaussian curvature to be negative, and if this problem were not addressed a saddle point would seem to have less fluctuation than an area that is completely flat. To fix this issue the absolute value of the curvature is taken. It is calculated in the following manner:

\[ G_i = \text{AbsoluteValue}((2\pi - \sum \theta_j)/A) \]

where \( G_i \) is the curvature of the vertex \( i \), \( \theta_j \) is the angle at that vertex in the j’th triangle, and \( A \) corresponds to a surface area. \( A \) is calculated by \( \sum A_j \) where \( A_j \) is the area spanned by a quadrilateral that consists of the circumcenter point of the j’th triangle, the vertex \( i \), and the two midpoints of the edges connected to the vertex. An illustration of this can be seen in Figure 6. Unfortunately, it might not be possible to accurately calculate the curvature given a single point: mesh smoothness and surrounding area needs to be taken into account. For example, a point may have high curvature only because of
3. Description of the Algorithms

Figure 7: The green circle is the vertex for which the curvature is being calculated. The yellow circles are the neighbours. The light blue area is $A_j$

noise. As a result, we might place a color source in a completely incorrect place. Therefore the curvature of a vertex is computed using the average of its own curvature and that of its neighbours. First the curvature for each vertex is computed, then the average is calculated.

3.4.2 Placing Sources

Before discussing the two placement algorithms, we explain two user-defined parameters that they have in common: minimum distance between sources and maximum number of color sources.

The first addresses the fact that it is undesirable to have 2 color sources extremely close to each other; the colors would simply blend together, making it pointless to have put 2 sources there in the first place. Therefore, the user chooses a minimum distance between any two sources. This is measured using the normalized distance for each source. This means it is possible for the minimum distance from A to be less than that from B in non-normalized space. At first glance this seems odd: why have a minimum distance that you can’t predict? The problem lies in the variability of stream surfaces. A surface that looks like a square is different from that of a vortex. Another problem is scale: if the user does not know the scale of the object then he is incapable of choosing an absolute distance. See Section 7.1.1 for alternatives.

The second addresses the placement of color sources in locations with undesirable curvature. The low (high) curvature method takes the vertex with the lowest (highest) curvature. Then it takes the next lowest (highest) that is at least the minimum distance away, and so forth until no more vertices fulfill the minimum distance criteria. As a result, it is possible that the low (high) curvature method ends up placing color sources in high (low) curvature locations. To counteract this the user can specify a maximum number of color sources. Though this might not be enough to solve the problem, it at least gives users a degree of control.
3.4 Curvature Based

Low Curvature Based

The idea behind putting color sources in low curvature areas is to place them where the surface is fairly flat over a relatively large area. Such an area would cover a fair amount of space on the screen from certain viewpoints. Giving only one color to such areas allows them to be visually distinguished. Unfortunately, simply because a vertex has low curvature does not mean that the surrounding area also has low curvature; it could simply be an exception - an area of lower curvature inside a larger area of high curvature. In addition, the mesh might be noisy. These issues are discussed in Section 3.4.4.

High Curvature Based

Two different methods were tried for high curvature points. The first, called compass based, places several different sources near the point. The other, called point based, places a single source at the point. The compass based approach was eventually abandoned, but I include a description of it here for completeness.

Compass Based

In this method the color sources are placed around the vertex, similar to a circle. Furthermore, they are as far away from each other as possible. This can be likened to a compass: the color source are the directions North, South, etc., and the high curvature vertex point would be the center of the compass. The color sources then color the surrounding area in different colors, making a convoluted area easier to comprehend. Because an area with high curvature could have multiple layers (example a spiral), the method would make the different layers have different colors.

Unfortunately, this method could not achieve the desired coloring result. The colors would either blend into one single color, and this could occur all across the mesh, making the entire stream surface monochromatic, or the color blending would be too artificially restricted and there would be a visual break on the mesh: For example the color would change from red to blue from one triangle to the next. While this would be comprehensible if there was only one layer, for multiple layers it appears as if the mesh were not connected in places it actually is. Therefore, I abandoned this approach.

Point Based

This approach is simply the high curvature analog of the low curvature method: each place receives one color source. This approach emphasizes high curvature areas, allowing the viewer to easily identify complex structures. Furthermore, because they have a different color from their surroundings, the viewer knows what belongs to the structure and what doesn’t. This method also shares some of the problems of the low curvature method, as discussed in Section 3.4.4.
3.4.3 Assigning Colors to Sources

After the sources have been chosen, I assign colors to them. The assignment of color is treated as a graph problem. The sources are the nodes, and there is an edge between two sources if they have a visible effect on or in the vicinity of each other. If for example two sources are on opposite sides of the mesh then they have little or no effect on each other, and there is no edge between them. In addition, the edges are weighted by the distance between the two connected sources. The optimal way of distributing the six colors is equivalent to finding a coloring of the graph where similar colors are assigned to sources connected by heavy or no edges. Unfortunately, graph coloring is NP-complete.

To approximate the solution I use a greedy algorithm and allow edges between all sources (i.e. a complete graph). By having a complete graph there is always an edge to follow to a source that has not yet received a color. I pick the first source and assign it a color. Then I pick the nearest (equivalent to the edge with the lowest weight) source without a color and assign it a color that is as different as possible from the previous one, and so forth, cycling through the different colors. If there are more sources than colors then this leads to certain problems, such as two sources with the same color being near each other. An example of this shown in Figure 8. Addressing this problem was outside the scope of this thesis.

Figure 8: An example of a bad placement of colors. The circle is the stream surface, the smaller colored circles indicate a color source. This example only uses three different colors as opposed to the 6 used in the algorithm.
3.4 Curvature Based

3.4.4 Curvature Method Comparison

Visually it is hard to tell which method produces which image. However, the high curvature version often does a better job of giving different colors to details, while the low curvature has a more consistent color for large flatter areas. Examples of low curvature and high curvature colorings are given in Figure 9 and Figure 10.

Unfortunately, these methods have several issues in common that are problematic. The first is the problem of placing a source in a location with low (high) local curvature surrounded by a larger area of high (low) curvature. The second problem is that the mesh itself might be noisy. While the effect of a noisy mesh can be reduced by taking the average curvature in an area, in extreme cases the mesh might need to be smoothed beforehand. Lastly, the methods are inherently sensitive: a small change in the parameters can result in massive differences in the coloring of the mesh. This can occur because the algorithm searches for the vertex with the next lowest (highest) curvature that is at least the minimum distance away. If the minimum distance is changed, another vertex may be chosen. This can affect later iterations, increasing the effect over time. See Figure 11 for an example in a low curvature case, and see Figure 12 for an example in a high curvature case.
Figure 9: Top image: no color. Middle image: low curvature. Bottom image: high curvature
Figure 10: Top image: no color. Middle image: low curvature. Bottom image: high curvature
Figure 11: Variation of minimum distance for the low curvature method and its effect.
From top to bottom: 0.3, 0.29, 0.28
Figure 12: Variation of minimum distance for the high curvature method and its effect. From top to bottom: 0.3, 0.29, 0.28
3.5 Grid Based

This approach attempts to find a viewpoint from which the most layers are seen and attempts to place the color sources accordingly. A viewpoint can be modeled as a position from which numerous rays are sent out, like raycasting. These rays intersect the stream surface. For each ray we count the number of layers it passes through. The ray with the most layers is the one along which we want to place the color sources.

The implementation consists of two steps. In the first step the mesh is analyzed in a style similar to rasterizing. In the second step the results from the first step are used to place the color sources.

3.5.1 Mesh Analysis

The user chooses a square grid of an arbitrary size (e.g. 1280*1280). The grid is then put in different positions around the mesh, and the mesh is then projected onto the grid. An example of this is seen in Figure 13. If a projected primitive (mesh-triangle) covers the center of a grid cell then the weight of that cell is increased, similar to rasterization. This simulates different camera positions around the grid searching for the view position with the most layers.

However, there are some factors that must be considered. For example, if the algorithm counts each primitive equally then a case like that in Figure 14, where a ray travels parallel to a surface area, might lead to a bad coloring of the mesh. It is therefore desirable that we only consider a primitive to be part of layer if a human would do so as well. This means each primitive must be considered depending on the viewpoint.

To simplify the calculation each primitive is considered individually. The weight of the primitive is calculated by taking 1 minus the dot product of the primitive’s normal with the view direction (both normalized). Primitives that are aligned with the view direction have little to no influence on a grid cell, while those perpendicular do. There are 2 possible ways to enhance this effect: only allow weights over a certain value as shown in Figure 15 or allow the weight to drop exponentially (e.g. use (1-cosine) to the power of five instead of just the cosine) as shown in Figure 16. The first requires the cutoff to be set fairly high before it takes effect. The higher the cutoff, the more likely the intersections between
3.5 Grid Based

Figure 14: In this image, if primitives were counted equally, these primitives would be considered as 6 layers, even though the ray is only traveling parallel to the layer.

Figure 15: Semi-hinge loss. With a normal hinge loss the line would reach 0 on the y axis before flattening out. In this thesis I simply ignore the values if they are below a certain value. If the cut-off value were deducted and the function clamped at 0, it would look like a hinge loss.

the view ray that maximizes the number of layers and the primitives will be closer to orthogonal. The second does have a tendency to change the result relatively easily as seen in Figure 18. If the result changes, the version with a higher power will have less color sources, but the intersections will be more orthogonal. This creates an obvious question: How high should the power be, or - in other words - in what cases would it be preferable to have the intersections closer to orthogonal? Unfortunately, the answer is mesh dependent and beyond the scope of this thesis.

3.5.2 Placing Color Sources

After we have analyzed the mesh according to the previous section, we know which grid cell of which viewpoint has the highest weight. We then use that grid cell to place the color sources. This is done by finding all the primitives that are rasterized to that grid cell. From each primitive we take one vertex and use it as a color source. To choose which color each source has, the vertices are sorted according to how far they are away from the viewpoint (imagine a ray being cast through the mesh - the vertices are sorted by how far along the ray their primitive is). Thus, we can simply iterate through the list of vertices and cycle through the colors.
3. Description of the Algorithms

3.5.3 Grid Implementation Details

A grid that acts like a viewpoint is moved on a sphere around the mesh. I attempt to have the positions distributed as evenly across the sphere. This is done by placing the grids along different latitudes of the sphere. The user defines the number of grids around the equator, as well as the number of different latitudes he wishes the grids to be on. On latitudes other than the equator I calculate the number of grids by using the cosine between the two latitudes and multiply it with the number of grids on the equator. This allows the distance between grids on the same latitude to be equal for all latitudes. It also allows the user to be fairly flexible with the number of grids he wishes to have. The grids do not have to cover the entire sphere, they only have to cover the northern or southern hemisphere. This is because a grid on the northern hemisphere has a corresponding grid on the southern hemisphere, and so the corresponding grid can be ignored.

For each grid, each mesh primitive is projected onto the grid. If it covers the center of a cell, then it’s weight is calculated and added to that cell. If the primitive does not cover any centers - which can occur if the grid resolution is low - it will not have any effect. It is possible that the object does not touch the edges of the each grid. The mesh is scaled according to its longest axis, i.e. it will only touch the edges of a grid if it is projected orthogonally to this axis.

An Nvidia graphic’s card was used for the implementation, and there are some changes to the grid data structure in order to parallelize the algorithm. Instead of using just one array for all primitives, I use as many arrays as I have Cuda blocks, and divide the primitives among them. For example, if I have 10 primitives and two blocks, I would assign primitives 1,3,5,7,9 to the first block, while the other primitives would be assigned to the second block. I use the term block-grid to refer to the array that belongs to a given block. In a second stage I divide up each of these block-grids into even smaller subgrids, like a chessboard, and assign a thread to each. Each thread only evaluates primitives that lie in its subgrid. That is, each thread takes each primitive that is assigned to its block and

![Figure 16: $1 - \cos(x)$ vs $(1 - \cos(x))^5$.](image)
Figure 17: From top to bottom are the different exponents and their effect on the result. The top has an exponent of 1, the middle 5, the bottom 10. The color sources are in the center of the colored areas. The areas are large so the reader can see them easily.
Figure 18: From top to bottom are the different exponents and their effect on the result. The top has an exponent of 1, the middle 5, the bottom 10. The position of the color sources can be seen in Figure 17.
checks to see if it lies in its subgrid. If it does, it rasterizes the primitive, otherwise it ignores it. The solution is obtained by adding up all the block-grids.

3.5.4 Grid Evaluation

Placing the color sources in such a manner has a simple benefit: in a more complex viewpoint, i.e., one with numerous layers, adjacent layers will not have the same color. However, if looked at from a different viewpoint, there may be adjacent layers that do not have different colors.

One problem is the amount of time required to rasterize the mesh on all the grids. To guarantee a good placement of the ray it may be necessary to have over one hundred grids, each with a fairly good resolution (from experience around 720*720). Another problem is the placement of color sources on certain meshes, resulting in a lack of colors in some areas. This occurs when a part of the mesh is far away from all color sources. An example of this is a square with vortices at each of its corners. Two of the vortices would be almost monochromatic.
4. Performance

4.1 System Specifications

The specifications of the system used for performing the measurements reported in this thesis are:

- CPU: Intel Core i7 CPU 930 @ 2.80GHz (4 CPUs)
- RAM: 12288MB
- Graphics Card: NVIDIA GeForce GTX 470

4.2 Curvature Methods

The CPU time for both curvature methods could not be measured because the program has multiple threads that are unrelated to the curvature calculation. This means that measuring the CPU time gives an incorrect result. Therefore, I measured the wall time. Because the measured wall time varied between runs, I measured it 6 times and took the average and standard deviation of the 5 shortest runs. The results are shown in Table 4.1. The algorithm would keep adding color sources until it couldn’t find any more places. The maximum number of color sources was 20, which in the experiments run here was never reached. The results indicate that the calculation takes around three seconds and is independent of the minimum distance required between two color sources. Furthermore, both methods are equally fast.

<table>
<thead>
<tr>
<th>curvature type</th>
<th>mindistance</th>
<th>average time (milliseconds)</th>
<th>standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>0.2</td>
<td>3076</td>
<td>618.8221</td>
</tr>
<tr>
<td>high</td>
<td>0.35</td>
<td>2252</td>
<td>217.5240</td>
</tr>
<tr>
<td>low</td>
<td>0.2</td>
<td>2708</td>
<td>458.0532</td>
</tr>
<tr>
<td>low</td>
<td>0.35</td>
<td>2908</td>
<td>217.5240</td>
</tr>
</tbody>
</table>

Table 4.1: Average time and standard deviation of curvature methods.

4.3 Grid Method

The largest factor in this method is the number of blocks used. If the rasterization for a triangle takes \( k \) steps on average, then the speed of the algorithm is \( O((n * k)/p) \) where \( n \) is the number of triangles, and \( p \) the number of threads. While increasing the number of subgrids can speed up the method, it only has a very small effect on the rasterization. It has a slightly larger effect when it comes to adding up all the blockgrids.

Unlike the case for curvature, the times for this algorithm did not vary substantially.
The times are shown in Figure 19. The resolution in this table indicates the resolution along an edge, e.g., 640 means that the grid is 640*640. I chose 720 and 1080, which are HD resolutions, and 640 and 1280, which are common monitor resolutions. The time is wall time and measured in milliseconds. Note that the time showed next to the axis has to be multiplied by $10^4$ as shown at the top. The circles and squares indicate the number of subgrids. Lines with squares have 16*16 subgrids, lines with circles have 8*8. The color indicates the number of blockgrids. Blue has 32, red has 24, and green has 16. A total of 249 viewpoints were used, and the mesh has 58741 triangles. The times include only the time to rasterize the mesh for all viewpoint grids.

It should be noted that it is likely that a more efficient algorithm exists because it is possible to render the meshes (without effects) at a significantly higher frame rate. Therefore a method based on the graphics pipeline should be faster. Due to time constraints and the restrictions of the framework in which the implementation was developed, it was not possible to implement it using the pipeline.
5. User Study

5.1 Intent

Perception involves interaction with a human viewer. This means that I can only evaluate the effectiveness of color on perception by studying how humans interact with images created with the algorithms specified in Chapter 3. In addition I need to compare these images with those without color. This requires that I have an objective criteria for measuring their performance. To this end different tasks were designed to understand in what way the coloring might help. For example, the coloring might help in following the surface in an arbitrary direction, but not help following the border. In essence these tasks define what it means to perceive a stream surface. The tasks are defined in Section 5.3 below.

5.2 Study Design

An online user study was carried out with 219 users. When a user started the study he was put into one of 8 groups. Each of these 8 groups has to solve one version of the user study. A version is a specific combination of coloring and meshes, an example of which is included in Appendix A. The tasks were the same across all groups, but each group had to solve the tasks on different images. The images differed based on mesh or coloring, including without any color. There were 4 different tasks as described below in Section 5.3. The user had as much time as he wanted to read the description of a task, but only 4 minutes to solve that task. The amount of time was fixed, and the user could not proceed to the next task until the 4 minutes had expired. Each task consisted of 4 or 6 questions, which the user had to solve. Each stream surface is shown twice. There are two tasks that get solved on the same surface. One stream surface is used in first and third tasks, and the second in the second and fourth task.

The setup is illustrated in Table 5.1.

<table>
<thead>
<tr>
<th>Task</th>
<th>View Point</th>
<th>High Curvature</th>
<th>Low Curvature</th>
<th>No Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>behind layer</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>nearest neighbour</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>follow border path</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>

Table 5.1: A and B correspond to the two meshes used. The groups are based on the coloring method and the meshes used.

5.3 Task Explanation

There are 4 different tasks. For all tasks the maximum achievable score is 1. A correctly answered question contributes a value corresponding to the number of questions for that
task. For example, if there are 6 questions, then the value received for a question is 1/6. In the task where the user had to follow the boundary the user had to mark one or more boxes. If he marked an incorrect box, he was penalized for that question. However he couldn’t go below 0 for a question. The tasks are described in more detail in the example user study in Appendix A.

**Find the layer**  The participant is given 6 points on silhouettes and boundaries and must specify the layer on which each point lies.

**Number of layers behind**  The participant is given 6 points together with the layer they lie on. He must then answer how many layers are behind the point. The points do not necessary lie on a silhouette or boundary.

**Nearest neighbour**  The participant is given four points and must indicate which points are nearest to them. The participant can move any way he wants along the stream surface.

**Follow the boundary**  The participant is given a start and end point on the boundary of the streamsurface and must follow the boundary from one to the other and mark which other points lie on the path between the two points. This is repeated for four different start and end point pairs.

5.4 Evaluation of Answers

Because the user study was online, it was possible that people got distracted or did not pay attention. Because it is not uncommon for people to have a lot of incorrect answers, it is almost impossible to detect such cases, so I worked under the assumption that they were too low in number to have an effect. Therefore I concentrated on culling participants who had not answered a fair amount of questions. These were easy to find: if the user didn’t answer the question, the question would be marked unanswered. If the participant had for any task not answered at least 30% of the questions, all his results were dismissed. This means he has to answer one or two questions per task, depending on the number of questions.

In order to evaluate statistical significance, I use a t-test to evaluate the significance between the means of two independent groups (without color and some coloring method). Statistical significance is determined by a combination of standard deviation, difference in means, group size, and effect size. There was no statistical significance found.

5.5 Results

The results of the study are shown in Table 5.2. For each task and coloring method - including no color at all - the average and standard deviation is listed.

The ranking of a coloring within a task is based on the average value. As can be seen
in the table the ranking of the coloring algorithms varies from task to task. For each color algorithm there is at least one task in which no-color is ranked higher. If the color algorithms had consistently outperformed no-color, then it would be possible to say there is a trend that coloring improves perception, even if there is no statistical significance. However, they do not. Therefore it is possible to say that coloring a stream surface does not improve perception.

<table>
<thead>
<tr>
<th>coloring</th>
<th>task</th>
<th>average</th>
<th>standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>grid</td>
<td>behind</td>
<td>0.40666667</td>
<td>0.23857726</td>
</tr>
<tr>
<td>highcurv</td>
<td>grid</td>
<td>0.33333334</td>
<td>0.21754335</td>
</tr>
<tr>
<td>lowcurv</td>
<td>highcurv</td>
<td>0.35185188</td>
<td>0.21833012</td>
</tr>
<tr>
<td>nocolor</td>
<td>lowcurv</td>
<td>0.34000003</td>
<td>0.22618379</td>
</tr>
<tr>
<td>grid</td>
<td>layer</td>
<td>0.3933333</td>
<td>0.20071875</td>
</tr>
<tr>
<td>highcurv</td>
<td>border</td>
<td>0.43827167</td>
<td>0.18784106</td>
</tr>
<tr>
<td>lowcurv</td>
<td>neighbour</td>
<td>0.4691358</td>
<td>0.19035982</td>
</tr>
<tr>
<td>nocolor</td>
<td>neighbour</td>
<td>0.42666668</td>
<td>0.19558232</td>
</tr>
<tr>
<td>grid</td>
<td>neighbour</td>
<td>0.38</td>
<td>0.27812228</td>
</tr>
<tr>
<td>highcurv</td>
<td>neighbour</td>
<td>0.33333334</td>
<td>0.26352313</td>
</tr>
<tr>
<td>lowcurv</td>
<td>border</td>
<td>0.30555555</td>
<td>0.26498315</td>
</tr>
<tr>
<td>nocolor</td>
<td>border</td>
<td>0.31</td>
<td>0.2749327</td>
</tr>
<tr>
<td>grid</td>
<td>border</td>
<td>0.5933333</td>
<td>0.31561175</td>
</tr>
<tr>
<td>highcurv</td>
<td>border</td>
<td>0.45524693</td>
<td>0.2704891</td>
</tr>
<tr>
<td>lowcurv</td>
<td>border</td>
<td>0.5216049</td>
<td>0.27850989</td>
</tr>
<tr>
<td>nocolor</td>
<td>border</td>
<td>0.47333333</td>
<td>0.28172818</td>
</tr>
</tbody>
</table>

Table 5.2: User study results
6. Conclusion

This thesis investigated the hypothesis that artificial coloring can help viewers perceive stream surfaces. As part of this I developed algorithms for deciding on and placing colors based on curvature and simulated viewpoints. I designed a user study to test these algorithms with respect to perception. The study presented users with several tasks that tested their perception on images created with these algorithms. In essence, these tasks indirectly define what it means to perceive a stream surface, see Sections 5.1 and 5.3. The results of the study, however, did not indicate that coloring significantly increases the ability of viewers to perceive stream surfaces. While the study was nonconclusive, it may be possible to design a better study that does show this, as we describe below in Section 6.1. On the other hand, the study can be seen as support for another conclusion, namely that color does not improve perception of transparent surfaces. This can have positive implications, which we describe in Section 6.2.

6.1 Study

The user study in Section 5 indicates that adding color to the stream surfaces does not lead to a substantial increase in perception. However, the structure of the study is such that one cannot rule out the possibility that color does indeed help. There are several reasons for this. First, the time required to solve each task was not taken into account; the participants had 4 minutes to solve a task, and had to wait until the time had expired. It is possible that users with the colored images solved the tasks faster than those without. Second, it was an online study. This means that it was possible for the users to be distracted while solving the questions, or they might not have taken the test seriously. As mentioned in Section 5.4 an attempt was made to cull these users. But the degree to which such problems affected the study is unknown. Third, the users did not know anything about the study except that it was to try out some new method. This means that they didn’t know what the colors could indicate. As a result, even if adding color does not help on a perceptual level, it might help on a logical level. If a person knows that a color is tied to a surface, he knows that any sudden change in the color indicates a discontinuity. An example of this is shown in the marked areas in Figure 20. Without color it might be possible to perceive a connection in the middle, i.e., that the surface continues. This is, however, not the case, and the color indicates this, but the number of such cases might not make it worthwhile.

6.2 Possible Implications

Although the study did not rule out that color does have a positive effect, it may still be the case that it doesn’t. This would agree with the points raised in Kramer et al. [4], where the perception of color and objects occurs separately in the initial stages of processing and are only combined later. In this sense, the study can be considered as a support for their argument. While this is negative when we try to perceive stream surfaces, it has positive implications in other areas, such as using color to impart additional information.
If color has no noticeable effect on human perception, then it is unlikely for there to be a “bad” placement of colors. This would allow the use of color to indicate, for example, the magnitude of the velocity of a stream surface or the relative potential of an equipotential surface in an electromagnetic field.
Figure 20: An illustration of perceiving surface continuity.
7. Future Work

7.1 Curvature Method

7.1.1 Distance Between Sources
In the curvature-based methods a normalized distance was used to determine how close another color source could be. However, this normalization was done for each source. This means that the non-normalized distance varies from source to source. This is undesirable because of its inconsistency. Therefore it would be advantageous if there was a consistent manner of measuring that is also mesh independent. One possibility might be based on the area of a stream surface. This was not attempted in this thesis because I only considered the possibility after I had started my user study. However it might not be possible to simply take the area of a surface. After all, the surface area grows quadratically with the boundary. Using the square root of the area might give better results.

7.1.2 Sensitivity to Parameters
As was illustrated, a small variation in the parameters can cause a massive difference in coloring. This makes it difficult to find parameters that give good results. Future work would focus on reducing the sensitivity, so that users could iteratively home in on a good coloring, rather than rolling dice.

7.1.3 Cutoff Value
Currently, the curvature algorithms continue until either there are no more vertices far away enough from a color source, or the maximum number of color sources has been reached. This means it is possible that color sources are placed in a position having the opposite curvature (e.g. high curvature method placing a source in a low curvature area). This could be stopped by adding a minimum or maximum curvature limit to the algorithm. For example, the low curvature algorithm would only place a source if the curvature is below 0.8. Because this value is mesh dependent, it is not easy to determine. Future work could explore methods to determine such a value for a mesh.

7.2 Grid Method

7.2.1 Multiple Rays
When using the grid-based method on some meshes it is possible that some large areas end up being monotonically colored. An example of this is Figure 21. While this may seem like an extreme example, the coloring isn’t that bad. The grid-based method fails more spectacularly in other situations. For example, let us assume a square with a vortex of some sort at each of its corners. The method would result in only two of these vortices receiving color (which two is irrelevant), and the other two being only monotonic in color.
Figure 21: For this mesh there are large monotonic areas

One possibility to remedy this would be to cast multiple rays through the mesh. These could be from the same grid, or from others. While this may seem simple at first glance, there are several problems. Assuming a second ray is found, it raises the question of which colors get assigned to which color sources. There are at least two possibilities in this regard: the color sources for two rays receive their colors independently of each other, or some attempt to coordinate the colors is made, such as giving two sources the same color if they are close enough to each other.

7.2.2 Interactivity

Because the method is dependant on a simulated viewpoint, the question of interactivity quickly arises. After all, if it might help from certain positions, would it not help more if we can also color the mesh optimally for every viewpoint? As I mentioned in Section 4.3 the method is not sufficiently fast to support interactivity. A large speed boost might be achievable if the graphics pipeline is used. For interactive use there is at least one other problem to surmount: there should be no sudden jumps in coloring when there is only a small change in perspective. Casting a single ray in the manner described earlier does not fulfill this requirement.
8. Bibliography


Appendix A

This section contains a copy of a version of the user study.
**Task: Find the number of layers behind a point**

In this task you are given a point and you must specify how many layers are directly behind this point. You are given the layer on which the point is located (e.g. A(L2) means that point A is on layer 2).

In this picture the **point A** is on the first layer (A(L1)) and there are **3 layers** behind it.

The point B is on the first layer (B(L1)) and there are **2 layers** behind it.

The point C is on the second layer (C(L2)) and there is **1 layer** behind it.
Task: Find the number of layers behind a point

Please specify the number of layers behind a point for each point. 0 is no layers, 1 is 1 layer and so on. Mark one item per line.

<table>
<thead>
<tr>
<th>Point</th>
<th>unsolved</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(L2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B(L3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C(L2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D(L2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E(L2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F(L3)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Task: Find out on which layer a point is located

For this task you are asked to specify on which layer a boundary or silhouette point is located. All points are on a boundary or silhouette. The front layer is L1.

<table>
<thead>
<tr>
<th>Image</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Image of A]</td>
<td>In this picture point A lies on the first layer, since there are no other surfaces in front of the marked boundary.</td>
</tr>
<tr>
<td>![Image of B]</td>
<td>In this picture, point B lies on the third layer, since there are two other layers in front of the marked boundary.</td>
</tr>
<tr>
<td>![Image of C]</td>
<td>In this picture, point C lies on the second layer, since there is one layer in front of the marked silhouette.</td>
</tr>
</tbody>
</table>

Important: All points are on a boundary or silhouette. There is only 1 boundary or silhouette at a point (though there may be another really close by). Only consider the layers that are DIRECTLY in front of the point itself.
**Task:** on which layer is the point located.

Please specify the layer for each point. L1 is the front-most surface layer, L2 the second layer, and so on. Cross one item per line. Only pay attention to the layers directly in front of the point.

<table>
<thead>
<tr>
<th>Point</th>
<th>unsolved</th>
<th>L1</th>
<th>L2</th>
<th>L3</th>
<th>L4</th>
<th>L5</th>
<th>L6</th>
<th>L7</th>
<th>L8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Task: find the nearest point when moving on a surface

For this task you are asked to follow the surface as if walking on it. You will be given one point on the surface and have to decide which of the other points is closest to it.

In this task, you will also be given the layer on which a point is located. Point A(L2) means that the point is on the second layer under the position labeled A. In this picture, this would be the back side of the smaller tube.

Similarly, point B(L1) is a point on the first surface layer at position B. In this picture this would be the front layer of the smaller tube.

Consider the points A(L2) and B(L1) in this picture. The shortest path along the surface between these points is highlighted blue.

In this picture, you are given the start point A(L2) and two candidates B(L1) and C(L2). Point B(L1) is the closest point to A(L2); both B(L1) and C(L2) lie on the front surface of the smaller tube, but the path from A(L2) to B(L1) is much shorter than the path from A(L2) to C(L2).
Appendix A

Task: find the nearest point when moving on a surface

Which of the five given points is closest to the start point? Mark one item per line.

<table>
<thead>
<tr>
<th>Point</th>
<th>unsolved</th>
<th>A(L1)</th>
<th>B(L2)</th>
<th>C(L3)</th>
<th>D(L2)</th>
<th>E(L2)</th>
<th>F(L1)</th>
<th>G(L2)</th>
<th>H(L2)</th>
<th>I(L2)</th>
<th>J(L2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A(L1)</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>B(L2)</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>E(L2)</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>J(L3)</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>
Task: follow a surface boundary

For this task you are asked to follow the boundary of a surface. You will be given two points on the boundary and have to decide which of the other points lie on the shortest path between the two points. It is possible that NO other points lie on the shortest path. Mark the box None in this case.

In this picture, you are given two end points A and B. The shortest path along the surface boundary between these points is highlighted blue.

In this picture, point C lies on the shortest path from A to B along the surface boundary.

In this picture, point D does NOT lie on the shortest path from A to B along the surface boundary, since the path from A to B over D is much longer than the path in the previous example.

In this picture, point D does NOT lie on the shortest path from A to B along the surface boundary, since D does not lie on a boundary at all.
Task: follow a surface boundary

Which of the following points lie on the shortest path between the given end points? For each row, check all items where the crosshair identifies a point on the shortest path along the surface boundary between the given start and end points.

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>None</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>F</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This section contains additional images of the different methods.
Figure 22: Rendering of a toroid. From top: no color, bottom: low curvature.
Figure 23: Rendering of a toroid. top: high curvature, bottom: grid.
Figure 24: Rendering of a toroid. From top: no color, bottom: low curvature.
Figure 25: Rendering of a toroid. top: high curvature, bottom: grid.
I declare that I did all the work myself.

3020 Riedbach, 24.03.2013

Nicholas Waldin