# A Visibility Algorithm for Converting 3D Meshes into Editable 2D Vector Graphics

Elmar Eisemann *
Saarland University / MPI

Sylvain Paris †
Adobe Systems, Inc.

Frédo Durand ‡
MIT CSAIL

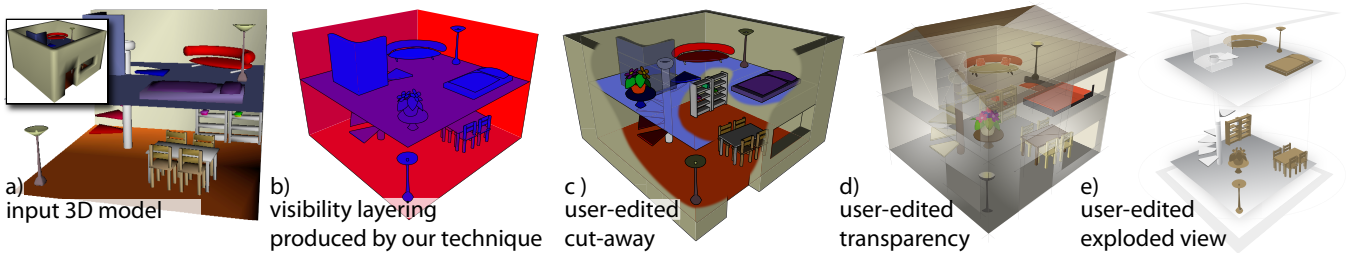


**Figure 1:** *Starting with a 3D model (a), we derive an editable layered vector representation (b). We introduce an algorithm to perform the necessary number of cuts to be able to spread arbitrary manifold models on layers in a consistent way. Layer representations are common to many vector illustration packages. Their power is illustrated in the various examples produced by us (c) and by artists (d,e) from our output.*

## Abstract

Artists often need to import and embellish 3D models coming from CAD-CAM into 2D vector graphics software to produce, e.g., brochures or manuals. Current automatic solutions tend to result, at best, in a 2D triangle soup and artists often have to trace over 3D renderings. We describe a method to convert 3D models into 2D layered vector illustrations that respect visibility and facilitate further editing. Our core contribution is a visibility method that can partition a mesh into large components that can be layered according to visibility. Because self-occluding objects and objects forming occlusion cycles cannot be represented by layers without being cut, we introduce a new cut algorithm that uses a graph representation of the mesh and curvature-aware geodesic distances.

**This paper is the author's version and only a preprint. The definitive version is published in the Transactions on Graphics 28(3) (Proceedings of SIGGRAPH 2009)**

**Keywords:** Visibility, vector graphics, NPR, geometry processing

## 1 Introduction

In a variety of contexts, artists need to create 2D vector art of a given 3D CAD-CAM model. The following examples, gathered from an informal survey of users of 2D vector graphics software, show the need for a 3D to 2D conversion enabling editing. For example, while architects model buildings and interiors in 3D, many presentations to clients are created by artists with 2D vector graphics. The 2D illustrator typically chooses the viewpoint that best suits the scene, converts the models into 2D illustrations, and shades them in a stylized manner characteristic of architectural previews. A similar pipeline is often used to create mechanical illustrations, e.g., cutaways and exploded views. In this case, illustrators also work on the layout of the diagram, increase the size of important small parts, remove useless details, and add annotations. In contrast, existing conversion methods are designed as the final step of the authoring pipeline, that is, they generate illustrations that are not meant to be edited. As a result, many artists need to perform the conversion manually, either tracing over a bitmap rendering, or working from the triangle soup produced by a simple projection. They spend tedious time recreating meaningful primitives from disconnected lines and organizing elements into depth layers [VectorTuts.com 2008]. While in some cases, it might be possible to create the final illustration directly from the 3D model, e.g. [Li et al. 2007; Li et al. 2008; Vollick et al. 2007], these dedicated methods cover only a subset of the tasks of an illustrator.

In this paper, we address the critical need to convert 3D meshes into editable 2D vector graphics, focusing on visibility and layering. Occlusions in vector illustrations are typically handled through ordering and layers that separate occluding and occluded objects. Our approach focuses on this aspect and produces a multi-layer illustration. The major difficulty of layer representations are self-occluding objects and complex occlusion dependencies. For instance, a self-occluding object always needs to be spread on several layers. Figure 1 shows such situations. The stairwell is passing through the floor and hence a cut is necessary. Even small details, like the lamp or some leafs of the flower contain self occlusions and need cuts. None of these elements could be placed on a single layer without causing order-related artifacts. Consequently, the mesh needs to be decomposed. The main contribution of this paper is how we decide which parts and along which line to cut. We describe a graph that represents the occlusion and adjacency relationships in the scene. We show that self-occlusions and occlusion cycles are *oriented cycles* of this graph, i.e., cycles containing at least one arc. Layer creation then boils down to a graph decomposition problem for which we describe an effective solution. Iteratively we will determine conflicts in the graph that we solve by cuts on the mesh. These cuts separate initially neighboring faces and hence remove cycles from the graph. We use biased geodesic distances over the mesh surface to locate these cuts so that the editing possibilities are preserved as much as possible.

*e-mail: eisemann@mmci.uni-saarland.de
†e-mail: sparis@adobe.com
‡e-mail: fredo@csail.mit.edu

We demonstrate our method on diverse inputs, from simple objects to complex scenes. We have asked artists to work using the documents produced by our algorithm and their results illustrate the possibilities offered by our method.

### 1.1 Related Work

A variety of techniques deal with 2D vector content [Isenberg et al. 2005] and we review some methods that are related to our work.

**Alternatives Pipelines** When possible, it can be beneficial to create the final illustration directly from the 3D model. For instance, Kalnins et al. [2002] directly specify NPR style on the 3D model in their WYSIWYG NPR approach, while Li et al. [2007; 2008] render cutaways and exploded views directly from 3D. In contrast, we address the case where artists need to work with 2D vector graphics software for flexibility or workflow reasons. From this perspective, our work is related to Asente's planar maps [2007]. However, we focus on the 3D-to-2D conversion and layer creation whereas the planar maps are flat 2D structures.

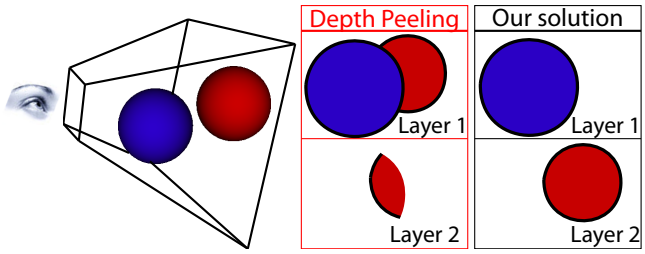


**Figure 2:** *Depth peeling gives correct layering, but breaks objects apart when it is not necessary, even in simple scenes. In contrast, our algorithm decomposes objects only when necessary.*

**Hidden Surface Removal and Transparency** Our method orders a 3D scene into a series of components sorted by depth. This relates to back-to-front rendering [Newell et al. 1972; Sutherland et al. 1974] and to depth peeling that is used to render transparent objects [Mammen 1989]. These techniques decompose a scene into elements that are sorted by depth so that rendering them in order yields a final image with the correct visibility or transparency. These algorithms work at the facet level and focus only on resolving occlusions. The input scene is broken up into many triangles and, from the illustrator's standpoint, the scene structure is lost. An example of this shortcoming is shown in Figure 2. Snyder and Lengyel [1998] proposed to avoid cutting by deriving blending functions to solve occlusion. Their solution used a similar, but simpler occlusion graph structure but did not resolve all situations. In contrast, we create valid layered documents meant for editing that respect the scene structure. Ryu [2001] also decomposes a model into occlusion-free layers. His algorithm works mostly at the triangle level and addresses rendering performance whereas we analyze a graph and focus on editing.

**Bitmap Vectorization** Bitmaps can be vectorized and many methods exist on this topic, e.g. [Orzan et al. 2008; Sun et al. 2007]. Commercial solutions, like SWIFT or LiveTrace render 3D models and vectorize these images. Some components, e.g., illumination, can be separated, but since all geometric primitives lie in the same layer, the input structure is lost and the resulting document is difficult to edit (Figure 3).

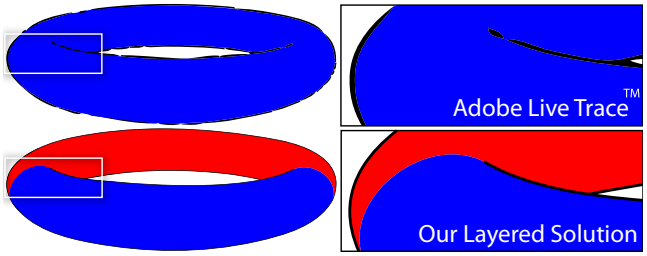


**Figure 3:** *Bitmap to vector conversion techniques such as LiveTrace (in Illustrator) lack accuracy and lack a layering structure.*

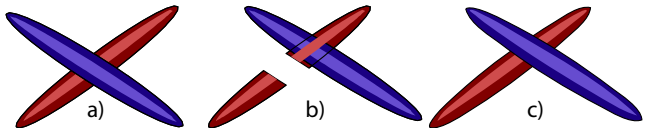


**Figure 4:** *Previous methods culled invisible parts for a fixed configuration (a). Manipulations reveal obvious artifacts (b). Our layering strategies support such modifications (c).*

**Vector Rendering of 3D Models** Recently, Stroila et al. [2008] and Eisemann et al. [2008] described algorithms to convert 3D models into vector graphics. But their objectives are significantly different. Stroila et al. focus on contour line extraction. Eisemann et al. handle conversion and stylization at the same time. Besides overlaid shading curves, the vector representations are flat. Editing is limited, as illustrated in Figure 4. Eisemann et al.'s algorithm relies on a planar map containing some information about hidden parts, but it is unclear how to extend it to layer decomposition.

### 1.2 Contributions

Our paper introduces the following contributions:

**Rendering a 3D model as a multi-layer vector illustration:** We transform an input 3D mesh into a 2D vector document that is organized in depth layers that respect the input structure.

**Graph-based mesh analysis:** We introduce an adjacency-occlusion graph that lets us study the visibility and connectivity of a 3D model on a compact and discrete structure.

**Cuts:** We decide where to cut a model using locally-defined distance functions on its surface.

## 2 Mesh Decomposition according to Visibility

**Overview** Given a manifold 3D mesh without self-intersection and a viewpoint, we create a layered 2D vector illustration. That is, we decompose the mesh into parts that can be ordered with respect to visibility, i.e., back to front rendering yields the correct image.

Furthermore, we seek to facilitate editing by the user. For this, we want to avoid polygon soups and aim for larger components.This is a major difference with the traditional Painter's algorithm [Newell et al. 1972] that operates at the polygon level. However, an object might exhibit self occlusion (Fig 5a), or a group of objects might not be possible to order according to visibility because of an occlusion cycle [Newell et al. 1972] (Fig. 6). In this case, we need to cut the mesh into multiple components to ensure visibility ordering. We want to avoid placing such cuts at places in the vector art where they might be difficult to handle for the user. In particular, we avoid the coincidence of cuts and contours (the separation between the front-

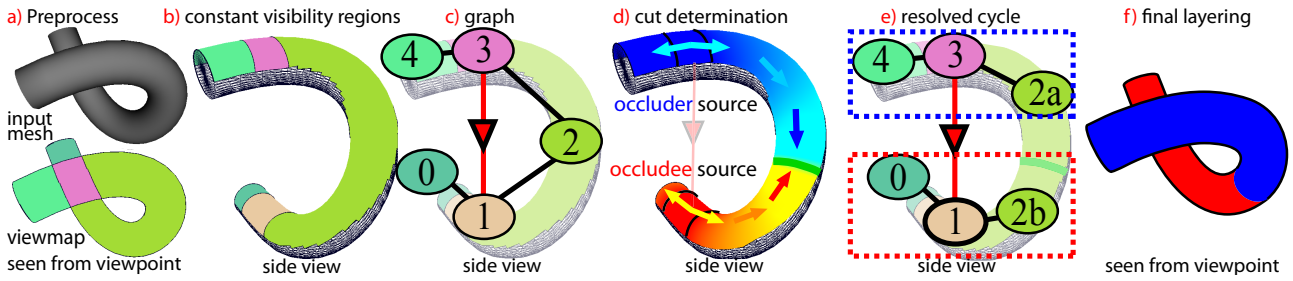


**Figure 5:** *Simple example of graph construction. We first remove the back faces (shown in wireframe on the middle views), then we compute the view map (a, bottom) by projecting the contours in the image plane. We backproject the view map cells to obtain regions of constant visibility (b). From these, we build a graph with regions as nodes, undirected edges between neighboring regions, arcs between occluding and occluded regions (c). Oriented cycles in this graph correspond to layering inconsistencies. The involved regions can be used to derive a distance field that helps us deduce the location of a necessary cut. The cut disconnects the two regions in conflict (e) and eliminates the cycle in the graph. Once all cycles are removed, a layering becomes possible (f).*

and backfacing surface) because it complicates element selection and navigation through the layers.

Figure 5 is an overview of our algorithm. We use two data structures to achieve our decomposition. A view map, e.g. [Grabli et al. 2004], decomposes the mesh into *regions of constant visibility*. These regions have a constant number of surfaces in front and behind them (Fig. 5b). In a sense, it is an over-segmented solution to our problem, just like depth peeling. By definition, these regions can be ordered back-to-front, but this strategy results in excessive subdivision and cuts perfectly aligned with contours. We want to create larger components and place cuts away from contours.

We also use a graph data structure, which we call the adjacency-occlusion graph. Its nodes are the regions of constant visibility of the mesh. The graph first represents adjacency between regions. In Figure 5 (c), the edges between regions 0 and 1, 1 and 2, etc., follow the mesh topology. The more of these edges we retain, the less disconnected regions will appear in the final output. We call regions/nodes *neighboring* if they are linked by an adjacency edge, and *adjacent* if there is a sequence of intermediate neighboring regions. In addition, the graph encodes information about visibility. A directed edge, the *occlusion arc* links regions that project to the same area of the view map. An edge goes from the region closest to the viewpoint to the occluded region, such as the edge between regions 3 and 1 in Figure 5 (c). These edges signify occlusions. Consequently, we call a node with incoming/outgoing occlusion arc(s) an *occludee/occluder*. We call nodes *connected* if there is a sequence of arcs (respecting orientation) and edges from one to the other. All cycles that include at least one such edge indicate that a proper visibility ordering is not possible and we refer to them as *oriented cycles*. Other cycles can exist and indicate adjacency on the mesh, which we try to maintain, where possible, in our output.

Given our adjacency-occlusion graph, our problem of mesh decomposition according to visibility is to eliminate oriented cycles in the graph. For this, we split nodes into subnodes, which corresponds to cutting the mesh across the corresponding region. Again, this is related to the Painter's algorithm where polygons need to be cut to eliminate oriented cycles, except that we deal with non-planar objects that can self occlude and hence, need to create a view map from contours. Finally, the precise cut location is based on a heuristic using a curvature-aware distance notion on the mesh (Figure 5(d)). In the end, we obtain a decomposition of the input mesh into a small number of components, two in Figure 5(e), that can be ordered back-to-front and yield correct visibility (f).

Our algorithm proceeds as follows, and is detailed below.

1. We remove the parts of the input mesh facing away from the input viewpoint.
2. We compute a *view map* of the input, that is, we project the mesh contours in the image plane of the input viewpoint. We backproject the view map in 3D to define the *regions of constant visibility* of the mesh.
3. We build a graph in which each region is a node and edges indicate either adjacency or occlusion between two regions.
4. We search for oriented cycles in the resulting graphs. These correspond to conflicts such as self-occlusions that require the mesh to be cut. We describe a method to treat several oriented cycles at the same time.
5. We cut the mesh to eliminate selected oriented cycles from the graph. The cuts are lines equidistant between occluding and occluded regions where the metric is based on the geodesic distance and the curvature of the mesh.
6. We assign a layer ID to the foremost component created by the cut. We extract it, update the graph and iterate Steps 4, 5, and 6 until all parts of the mesh are simplified on layers.

### 2.1 View Map

To capture the changes of visibility, we compute a *view map* by projecting all contours in the image plane of the viewpoint. The view map consists of cells delineated by projected contours. We backproject these cells onto the mesh to define *regions of constant visibility*. To gain more intuition, we observe a simple case: a point $a$ occluding another point $b$. Since $a$ occludes $b$, the two points are on the same line of sight and project on the same point in the view map. If we backproject the cell containing this point, we obtain two regions $A$ and $B$ that contain $a$ and $b$ respectively. By definition of the view map, there is no contour in $A$ nor $B$, which means that the entire region $A$ occludes the entire region $B$. This shows the usefulness of the decomposition: we can solve occlusions with regions since there are no visibility changes within them. The view map is related to Appel's quantitative invisibility [1967] that counts the objects in front of a point. However, our representation is not equivalent since we also take the objects behind into account.

**Building the View Map** We construct the view map and the constant visibility regions with a method inspired by BSP-trees. First, we detect the contours of the model. For each contour segment, we compute the equation of the plane containing this segment and the viewpoint. If a face intersects this plane, we compute the equation

of the plane containing this face and project the contour segment onto it. If this projection intersects the face, we cut the face along the contour plane. During this process, we make sure that the mesh remains manifold by maintaining mesh connectivity and subdividing the uncut faces neighboring to a cut face. Robustness of the cutting process is addressed by using a small $\epsilon$ for the plane intersection test, and by carefully treating vertices with a distance to the contour plane below this threshold (cf. supplemental material).

Regions are constructed with a flood-fill algorithm. Region boundaries are necessarily cuts or contours, so we assign a region ID to an unlabeled face and propagate it until we reach a cut or boundary. We then start a new propagation with a new unlabeled face and ID.

### 2.2 Adjacency-Occlusion Graph

Our adjacency-occlusion graph summarizes adjacency and occlusion relationships between regions. Each region of constant visibility is a node in our graph. If two regions are neighboring, we create an undirected *adjacency edge* between them. When possible, to preserve the model's structure, they should be assigned to the same layer. Occlusion arcs are added using ray tracing. A ray is shot through each region center and we add arcs between successively intersected region pairs. When creating layers, we must ensure that nodes linked by an occlusion arc are in different layers.

**Simple Example** The input model in Figure 5 is a bent cylinder that projects into an "alpha". An occlusion occurs where the cylinder passes on top of itself. The view map has 4 cells: the two ends, the center, and the overlapping part. Backprojecting the view map generates 5 regions, one for each cell, except for the overlapping cell which gives an occluder-occludee pair. We number the regions from 0 to 4. The undirected edges between neighboring regions create the chain $0-1-2-3-4$. The arc $3 \rightarrow 1$ represents the occluder-occludee pair. We cannot put the object in a single layer without overlap because it is self-occluding. The oriented cycle $1-2-3 \rightarrow 1$ indicates a conflict since it means that 1, 2, and 3 are in a connected component that contains an occlusion. The following section discusses such cases in more detail.

### 2.3 Finding Conflicts using the Graph

There are two main reasons to cut the input mesh. If an object occludes itself, we need to separate occluder and occludee by breaking the connectivity. The other case is an occlusion cycle such as the three bars of Figure 6 that cannot be ordered by depth. In our graph, both cases correspond to the same configuration: an oriented cycle.

If there are no oriented cycles in the graph, layering is possible without cutting the model. Each mesh component, defined by adjacency, can be put on one layer. The partial ordering induced by the occlusion arcs is used to sort the layers.

If there are oriented cycles, the input model has self-occlusions or occlusion cycles. We resolve these cases with cuts. In graph terms, a cut splits a node of an oriented cycle into subnodes. The subnodes are relinked in the graph by adding an adjacency edge between nodes whose corresponding regions on the mesh are neighboring and not separated by a cut. Formally, our objective is to split nodes until all oriented cycles are eliminated. For the bent cylinder (Figure 5), we cut the node 2. The subnodes $2a$ and $2b$ are not linked but $2a$ is linked to 1 and $2b$ to 3 because they are neighboring. After the cut, we have two layers: $\{0, 1, 2a\}$ and $\{2b, 3, 4\}$.

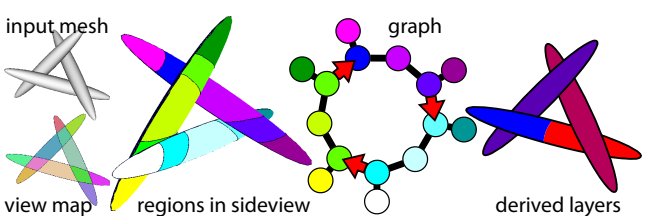


**Figure 6:** *Inconsistencies with respect to layers do arise from self occlusions, and occlusion cycles produced by chains of objects.*

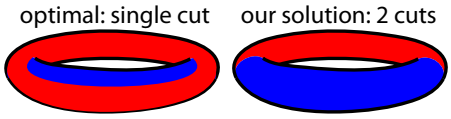


**Figure 7:** *Minimizing the number of cuts can lead to unnatural decompositions (left). In comparison, our heuristic yields more satisfying layers (right).*

The problem is related to the minimum-feedback-arc-set and minimum-feedback-vertex-set problems that seek to remove a minimal set of edges or nodes so that the remaining graph is acyclic. These graph problems are known to be NP-hard [Kann 1992]. Our problem is not equivalent because we do not remove nodes but split them and we use the input geometry to determine the graph adjacency after a cut. Also, the notion of optimality is hard to define in our case. Cut or region minimization can yield difficult-to-edit shapes (see Figure 7). This motivated our use of heuristics to create editable layers. Figure 8 summarizes our approach.

**function** CREATELAYERS(input: Graph $G$, Mesh $M$)

0. Extract the strongly connected components of $G$.
**for each** strongly connected component $g$ of $G$
1. Choose node $i$ with largest number of occlusion arcs in $g$.
2. Create cut mesh $M'$ from $M$ to break $i$'s oriented cycles:
   (a) Create a set $R$ containing $i$ and all adjacent occluders not neighboring to an occludee.
   (b) On Mesh $M$: Compute the equidistant curve $C$ between $R$ and all occluded regions.
   (c) Cut regions traversed by $C$, producing mesh $M'$. Split corresponding nodes in $g$ and update adjacency in $g$.
3. Create a layer containing $i$:
   (a) Create selection $L$ with $i$ and its adjacent nodes in $g$.
   (b) Assign a layer ID to $L$ and remove $L$'s nodes from $g$.
4. If $g$ is not empty, call CREATELAYERS($g$,$M'$).

**Figure 8:** *Pseudo-code for creating layers.*

Since we are concerned about cycles, we can restrict ourselves to *strongly connected components* which are maximal sets of nodes such that any two nodes are connected. Since all nodes of a cycle are connected, they are always contained within a single such component. Working within components is an optimization that lets us deal with smaller graphs (Step 0). We first process the region that occludes the largest number of occludees (Step 1). This heuristic is based on the intuition that this region is involved in many cycles since it occludes many regions. This strategy is similar to the heuristic proposed by Skiena [1998] to find feedback sets. This region has no occluder and no region has more occludees, hence, it also has the maximum difference between the number of outgoing edges (occludees) and the number of incoming edges (occluders), which is Skiena's criterion. Once we have selected the most occluding region, we search for other occluders to process simultaneously (Step 2a). Figure 9 shows that handling occluders one by one

can produce unsatisfying layers. We select all those occluders not next to an occludee to prevent superimposed splines due to a cut aligned with a contour. We compute a cut line (as detailed in the next section) to separate these occluding regions from the occludees (Step 2b). We split the crossed faces and update the graph accordingly, that is, no edges across cuts and undirected adjacency edges for neighboring regions on the mesh (Step 2c). We use a flood-fill algorithm to extract the portion of the model delineated by the cut that contains the most occluding region selected earlier (Step 3a). We assign a new layer ID to the corresponding mesh faces and remove the regions from the graph (Step 3b). If faces remain in the current component, we start the layer creation algorithm again on the remaining graph (Step 4).

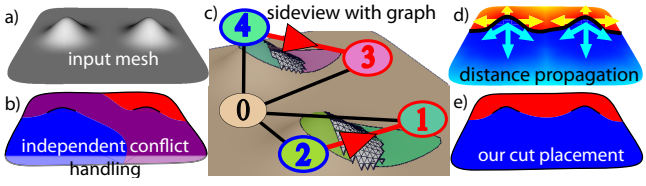


**Figure 9:** *Two bumps (a) already lead to an interesting layering problem. Treating conflicts in the graph (c) independently leads to undesirable cuts (b). Our diffusion solution (d, explained in Section 2.4) treats many conflicts at once to improve the cuts (e).*

### 2.4 Cutting the Model

Once we have selected the regions to separate, we cut the model. To gain intuition, we first consider computing the geodesic distance from both the occluding and the occluded region sets. We locate the equidistant points, and we cut the model along this boundary. The cut separates the occluder and occludee and breaks the cycles that ran through its boundary.

The distance on the mesh is approximated by shortest paths along edges. We compute the distance of every vertex to the nearest occluding and occluded regions using a connectivity-respecting front-propagation algorithm based on a priority queue. Each vertex stores a marker indicating whether the closest region is an occluder or an occludee. The cut is placed where the two fronts meet. Most models yielded satisfying cuts despite the path approximation. For regular quad meshes random triangulation and subdivision helped.

The geodesic distance may not always yield satisfying lines. The problem of finding "good lines" on 3D models has been largely studied in the context of mesh segmentation, e.g. [Attene et al. 2006], and for line drawings, e.g. [DeCarlo et al. 2003; Judd et al. 2007; Cole et al. 2008]. Several options have been proposed and one or another performs better depending on the considered model.

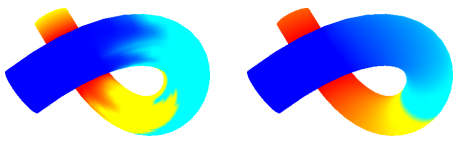

**Figure 10:** *Penalizing curvature often leads to fuzzy boundaries (left), the opposite more quickly isolates the part in conflict and propagates a frontier that respects the feature (right).*

Our strategy is to weight the metric in the direction $\vec{d}$ at point $x$ by a factor $w(x, \vec{d})$. We consider two options. The first one places more weight on high curvature points: $w_\times(x, \vec{d}) = |\kappa_n(x, \vec{d})|$ where $\kappa_n$ denotes the normal curvature at $x$ in direction $\vec{v}$. The second one weights them less:

$$w_\div(x, \vec{d}) = \frac{1}{1 + |\kappa_n(x, \vec{d})|} \quad (1)$$

In our tests, the latter performed better and our results are obtained with it. This comes from the fact that self-occluding areas are often thin and elongated akin to the bent cylinder (Fig. 5). The weight $w_\times$ exaggerates the distances along the feature, which creates wiggly cuts. In contrast, $w_\div$ attenuates the distance along the features, which generates smoother boundaries (Figure 10).

Contours induce layer boundaries. At cusps, our smooth cut separates tangentially meeting conflict regions. Because of how the diffusion is initialized, the cuts must join up with contours on the surface at obtuse angles. Since the surface is also viewed edge on at these points, these cuts extend contours with $G^1$ continuity in 2D [DeCarlo et al. 2003]. In general, the distance diffusion process leads to a smoothly delineated extracted layer (Figure 11).

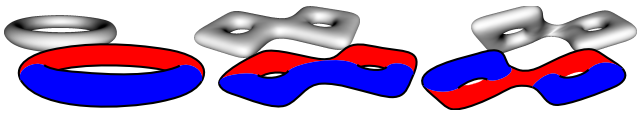

**Figure 11:** *Simple cases showing the cut-placement behavior.*

### 2.5 Creating the Final Output

After a cut, the extracted faces have no occlusion and can be flattened on a layer. Each such face set receives a layer ID. After all cuts, the graph is acyclic and we can sort the layers by depth using topological order. As a post-process, we further reduce the number of layers. Each layer receives a new ID equal to the maximum ID of its occluders plus one. Hence, no occluder-occludee pair is assigned the same ID, while layers sharing the same ID can be merged.

To create the vector representation of a layer, we extract its boundary by walking over the faces. We obtain a three-dimensional polyline. This boundary is projected in 2D and we apply a Newton-Raphson optimization [Press et al. 1992] to convert the obtained 2D polyline into splines. To prevent holes between neighboring regions, we decompose this 2D polyline into sub-chains such that each sub-chain is of maximal length and the faces on the other side have the same layer ID. We convert the sub-chains into splines independently. This ensures that the boundary between two regions is represented by the same spline.

To support objects with holes such as tori, all splines are grouped in a single compound path. The interior and exterior regions are then correctly defined by the winding number. This feature is common in vector editing packages. We also found it useful to output separate splines for contours to ease the application of a style.

## 3 Results

**Performance** We have tested our algorithm with a number of polygonal models. Performance statistics can be found in Table 2. Our prototype is able to convert 3D models of moderate complexity into editable 2D vector graphics. Computation costs are dominated by ray casting because we do not use any acceleration structures. Note the relatively low number of final layers, which is important to facilitate the tasks of artists editing the 2D vector document.

One limitation of our approach are poorly-conformed 3D meshes represented as polygonal soups. Mesh repair techniques,

| | nodes | occluders | occludees | both | edges | directed | SCC | largest SCC | cut executions |
|---|---|---|---|---|---|---|---|---|---|
| Apartment (Fig. 1) | 4,825 | 1,122 | 1,308 | 2,336 | 19,834 | 9,646 | 253 | 74 | 4 |
| Elephant (Fig. 13) | 37 | 15 | 15 | 1 | 59 | 17 | 4 | 26 | 2 |
| Car (Fig. 15) | 1,234 | 425 | 390 | 271 | 3,565 | 1,329 | 92 | 473 | 16 |
| Alien (Fig. 16) | 150 | 60 | 58 | 9 | 250 | 85 | 28 | 32 | 8 |

**Table 1:** *Statistics about the numbers of nodes (total, occluding, occluded, occluding & occluded), number of edges (total, directed), number of strongly connected components (SCC), number of nodes in the largest SCC, number of executions of the cut algorithm.*

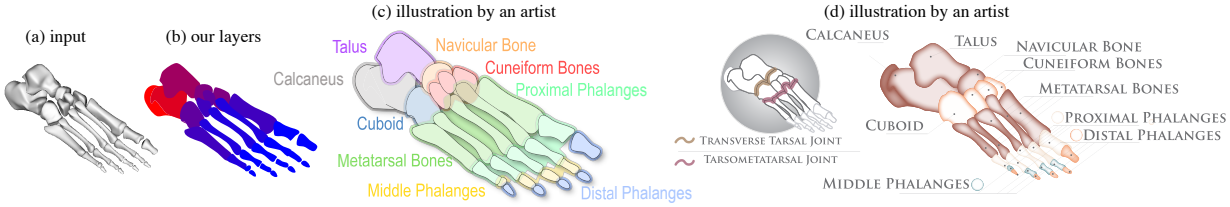


**Figure 12:** *The original model (a) was transformed into a layer representation (b). The output was then transformed by artists. For the example (c), the original input was rich enough to finish the entire illustration in about 15 minutes.*

| scene | apartment | elephant | car |
|---|---|---|---|
| number of faces | 20,427 | 38,399 | 65,192 |
| cycle removal and layering | 3.3s | 0.3s | 1.6s |
| graph creation | 84s | 11s | 154s |
| final # of visibility layers | 22 | 3 | 15 |

**Table 2:** *Performance on a Pentium Centrino 2.2 GHz. The graph creation dominates the computation due to missing optimizations.*

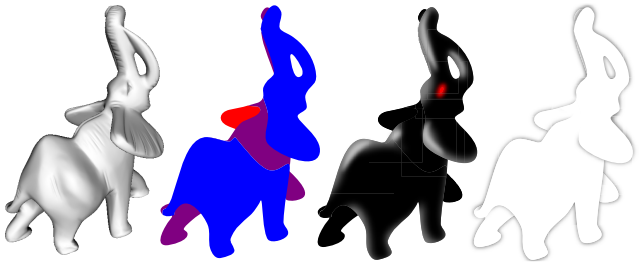

**Figure 13:** *Examples. Stylization took 5 to 10 minutes.*

e.g., [Nooruddin and Turk 2003] could be used as a preprocess to our algorithm to ensure mesh connectivity.

**Prospective-User Feedback** We asked a few illustrators to work with the documents that we created. The illustrators were recruited from research participation mailing lists and received small compensation for their time. They worked with the editing software Illustrator. We asked them to modify the documents along guidelines such as "create an educational illustration" or "create an architectural drawing with cutaways" (Examples are given as supplemental material). The participants were not supervised nor observed while they drew. Beside the theme of the illustration, the illustrators were free to draw what they wanted. The participants answered a short questionnaire about their experience.

Figures 1, 12, 14, 15 show returned illustrations. The edits range from simple changes of color and shape (Fig. 14) to complex layout design (Fig. 1) and sophisticated inclusion of bitmap elements (Fig. 15). The level of complexity of some of these edits demonstrate that our results are usable for advanced editing tasks. In addition, the received comments were globally positive as shown by

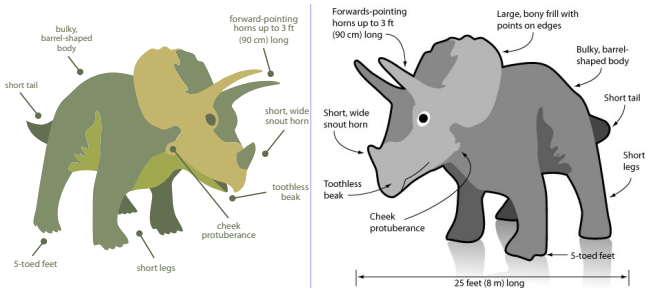


**Figure 14:** *For this example, professional artists were asked to create an annotated illustration as can be found in schoolbooks.*

the following examples.

*"The layers are very handy in order to quickly select and modify groups of shapes. I like this arrangement. Nice and flexible - it matches my usual work flow in the use of layers - especially when working with content from 3D. It's also nice to have ordered layers. It would be nice to see options for generating this content."*

*"[The layers] were helpful. Made Creating depth easier. You don't have to be as exact since you can use existing element to "trap" or Clip objects placed below them."*

*"[The layers] were helpful to distinguish between the various sections of the figure. I personally do not rely heavily on layers, so it was more of an adjustment for me to use them while working. I understand the benefits of them, but I never got into the habit of using them. Old habits die hard, I guess."*

This last comment is particularly interesting as it describes a usage of our layers that we did not expect. We also received less positive comments, mostly related to the usage of Illustrator. For instance, a participant did not know how to reduce the number of control points, which slowed them down. The main criticism by one participant was the number of layers that can be large with complex illustrations such as the apartment (Fig.1). This suggests creating hierarchical layers as future work. An interesting observation is that none of the five artists commented on the positioning of the cut lines, showing that our motivation to make areas quickly accessible by avoiding cuts on contours seems to be a valid criterion.

## 4 Summary

We presented a method to convert 3D models into editable vector graphics. Our approach generates documents structured in visibility layers. An informal user study showed that these documents address a need and help illustrators create compelling results. Even though, we focused on static images, our method can also be helpful for 2D animation, e.g., Adobe's Flash (see video and Figure 16).

*We thank Maneesh Agrawala, David Macy, the MIT Graphics Group, the Adobe Creative Technologies Lab, and the reviewers for their insightful comments, Tim Toy for organizing the user study, Hedlena Bezerra, Lionel Baboud, and Robert Herzog for helping with the text, Aim-At-Shape, ViewPoint DataLabs, Dassault Système SolidWorks Corp. for the models. Frédo Durand acknowledges a Microsoft Research New Faculty Fellowship, and a Sloan Fellowship, and a generous gift from Adobe. This work was supported by the Cluster of Excellence on Multimodal Computing and Interaction - www.m2ci.org.*

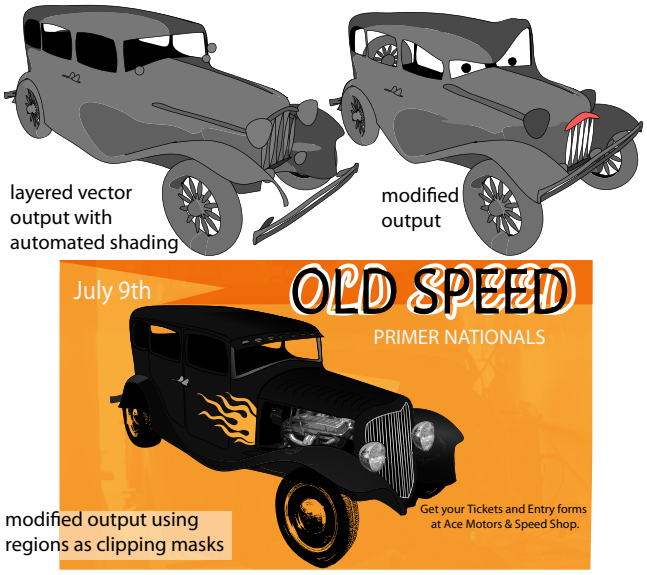


**Figure 15:** *The upper left illustration was created automatically. The upper right shows a version edited by one of the authors, which took about 20 minutes including the conception of the design. Note that we were able to reveal the back wheel that was hidden in the initial view. The bottom illustration was done by a professional artist who used the extracted regions to clip bitmap components.*

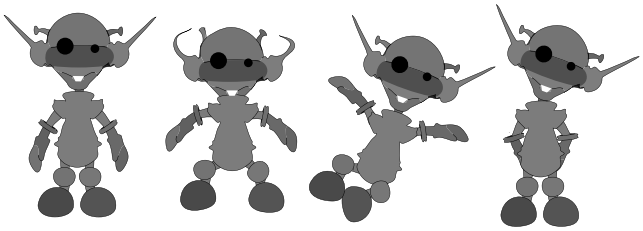

**Figure 16:** *Simple 2D vector animation using the initial pose (left).*

## References

APPEL, A. 1967. The notion of quantitative invisibility and the machine rendering of solids. In *Proceedings of the 22nd National Conference*, 387–393.

ASENTE, P., SCHUSTER, M., AND PETTIT, T. 2007. Dynamic planar map illustration. *ACM Transactions on Graphics 26*, 3. Proceedings of ACM SIGGRAPH.

ATTENE, M., KATZ, S., MORTARA, M., PATANÉ, G., SPAGNUOLO, M., AND TAL, A. 2006. Mesh segmentation: A comparative study. In *Proceedings of Shape Modelling International*.

COLE, F., GOLOVINSKIY, A., LIMPAECHER, A., BARROS, H. S., FINKELSTEIN, A., FUNKHOUSER, T., AND RUSINKIEWICZ, S. 2008. Where do people draw lines? *ACM Transactions on Graphics 27*, 3. Proceedings of ACM SIGGRAPH.

DECARLO, D., FINKELSTEIN, A., RUSINKIEWICZ, S., AND SANTELLA, A. 2003. Suggestive contours for conveying shape. *ACM Transactions on Graphics 22*, 3. Proceedings of ACM SIGGRAPH.

EISEMANN, E., WINNEMÖLLER, H., HART, J. C., AND SALESIN, D. 2008. Stylized vector art from 3d models with region support. *Computer Graphics Forum 27*, 4. Proceedings of the Eurographics Symposium on Rendering.

GRABLI, S., TURQUIN, E., DURAND, F., AND SILLION, F. 2004. Programmable style for npr line drawing. In *Rendering Techniques (Proceedings of the Eurographics Symposium on Rendering)*, ACM Press, 33–44.

ISENBERG, T., CARPENDALE, M. S. T., AND SOUSA, M. C. 2005. Breaking the pixel barrier. In *Proceedings of the Eurographics Workshop on Computational Aesthetics in Graphics, Visualization and Imaging*.

JUDD, T., DURAND, F., AND ADELSON, E. H. 2007. Apparent ridges for line drawing. *ACM Transactions on Graphics 26*, 3. Proceedings of ACM SIGGRAPH.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: Drawing strokes directly on 3D models. *ACM Transactions on Graphics 21*, 3. Proceedings of ACM SIGGRAPH.

KANN, V. 1992. *On the Approximability of NP-complete Optimization Problems*. PhD thesis, Royal Institute of Technology, Stockholm.

LI, W., RITTER, L., AGRAWALA, M., CURLESS, B., AND SALESIN, D. 2007. Interactive cutaway illustrations of complex 3D models. *ACM Transactions on Graphics 26*, 3. Proceedings of ACM SIGGRAPH.

LI, W., AGRAWALA, M., CURLESS, B., AND SALESIN, D. 2008. Automated generation of interactive 3D exploded view diagrams. *ACM Transactions on Graphics 27*, 3. Proceedings of ACM SIGGRAPH.

MAMMEN, A. 1989. Transparency and antialiasing algorithms implemented with the virtual pixel maps technique. *Computer Graphics and Applications 9*, 4.

NEWELL, M. E., NEWELL, R. G., AND SANCHA, T. L. 1972. A solution to the hidden surface problem. In *Proceedings of the ACM National Conference*.

NOORUDDIN, F. S., AND TURK, G. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE Trans. Vis. Comput. Graph 9*, 2, 191–205.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: A

vector representation for smooth-shaded images. *ACM Transactions on Graphics 27*, 3. Proceedings of ACM SIGGRAPH.

PRESS, W., TEUKOLSKY, S., VETTERLING, W., AND FLANNERY, B. 1992. *Numerical Recipes in C*. Cambridge University Press, Cambridge, UK.

RYU, D. 2001. *Visibility Layer Decomposition*. Senior thesis, Harvard University.

SKIENA, S. S. 1998. *The Algorithm Design Manual*. Springer-Verlag.

SNYDER, J., AND LENGYEL, J. 1998. Visibility sorting and compositing without splitting for image layer decompositions. In *Proceedings of ACM SIGGRAPH*.

STROILA, M., EISEMANN, E., AND HART, J. C. 2008. Clip art rendering of smooth isosurfaces. *IEEE Transactions on Visualization and Computer Graphics 14*, 1.

SUN, J., LIANG, L., WEN, F., AND SHUM, H.-Y. 2007. Image vectorization using optimized gradient meshes. Proceedings of ACM SIGGRAPH.

SUTHERLAND, I. E., SPROULL, R. F., AND ROBERT, A. S. 1974. A characterization of ten hidden-surface algorithms. *ACM Computing Surveys*.

VECTORTUTS.COM, 2008. Rendering a 2d spark plug diagram from 3d components in illustrator. `http://vectortuts.com/tutorials/illustration/`.

VOLLICK, I., VOGEL, D., AGRAWALA, M., AND HERTZMANN, A. 2007. Specifying label layout styles by example. In *Proc. of ACM Symposium on User Interface Software and Technology*.