Computer Graphics International 2004 (CGI), June 16–19, Crete, Greece. IEEE Computer Society Press.

Consistent Normal Orientation for Polygonal Meshes

Pavel Borodin Gabriel Zachmann Reinhard Klein

Institute of Computer Science II, University of Bonn, Germany
{borodin, zach, rk}@cs.uni-bonn.de

Abstract

In this paper, we propose a new method that can consistently orient all normals of any mesh (if at all possible), while ensuring that most polygons are seen with their front-faces from most viewpoints. Our algorithm combines the proximity-based *with a new* visibility-based *approach. Thus, it virtually eliminates the problems of proximity-based approaches, while avoiding the limitations of previous solid-based approaches.*

Our new method builds a connectivity graph of the patches of the model, which encodes the "proximity" of neighboring patches. In addition, it augments this graph with two visibility coefficients for each patch. Based on this graph, a global consistent orientation of all patches is quickly found by a greedy optimization.

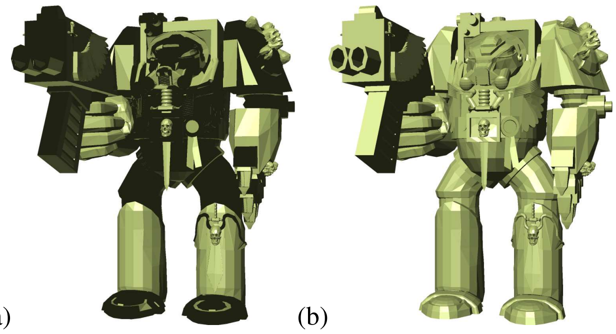
We have tested our new method with a large suite of models, many of which from the automotive industry. The results show that almost all models can be oriented consistently and sensibly using our new algorithm.

1. Introduction

Boundary representations consist of a set of primitives, with or without topological information. Important examples of such primitives are polygons, NURBS patches, and patches generated from subdivision surfaces. In many areas of computer graphics, it is desirable or even necessary that the primitives of a model be consistently oriented (in a sense to be defined later), i.e., that the normals point in the "correct" direction.

One area, where consistent normal orientation is highly desirable, is real-time rendering. Figure 1a shows an example of an inconsistently oriented polygonal model, which results in incorrect lighting (the so-called "checkerboard effect"). This could be remedied by two-sided lighting, or by doubling the number of light sources. However, both ways will decrease rendering performance. In addition, correct normals are still needed to perform back-face culling, a technique to further improve rendering performance.

Similarly, correct orientation of a model's primitives is important in ray tracing and radiosity, otherwise lighting artifacts will be caused.



(a) (b)

Figure 1. (a) A model consisting of 2398 separate surface patches with inconsistent orientation; due to the lighting, the faces oriented "backwards" are rendered dark. (b) Same model after applying our algorithm (with exactly the same lighting).

More importantly, inconsistent normal orientation can be fatal for many well-established mesh processing algorithms. For example, the mesh simplification algorithm proposed by Garland and Heckbert [4] uses vertex normals to determine the order in which contraction operations are to be performed. If applied to a model with inconsistent normals, this algorithm will produce severe artifacts.

Other areas are the computation of basic object properties, such as volume and mass, rapid prototyping [11], NC machining, and the optimization of wireless communication systems [7].

Unfortunately, most modelling tools, in particular CAD tools, pay little attention to consistent normal orientation. There is no feature that automatically orients the normals, so designers have to manually orient each patch.[1] Many models are not designed as solids, but just as a single sheet of patches (such as a windshield). In addition, many models contain other geometric flaws, such as unintentionally intersecting primitives, cracks or gaps, and T-junctions.

In this paper, we present a new algorithm for consistently orienting the normals of a boundary representation, even in the presence of gaps, T-junctions, and intersections. The input consists of an arbitrary set of primitives, without any topology information. The algorithm can handle non-closed

[1] At many German automotive companies, there are design guidelines that include rules how designers should orient surface patches, but it is often very difficult to enforce them.

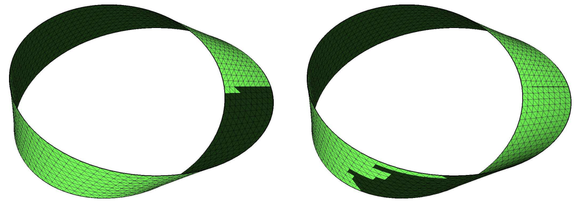


Figure 2. Some models cannot be oriented consistently. The Möbius strip is a simple example. Here, there are different possible solutions.

and even non-manifold objects. Figure 1b shows the output of our algorithm for the model from Figure 1a, which has consistent normals everywhere.

Here, we will describe our algorithm for polygonal objects. Note, however, that it works just the same for objects consisting of NURBS or other primitives. Thus, tessellations from such models with consistent normals would also be consistent.

Our algorithm first divides the model into a set of manifold surface patches and consistently orients the polygons within each patch. Then, it determines the *closeness* of the patches to each other across their common boundaries, and it approximates the *visibility* for each patch regarding all possible viewpoints. Based on these it orients the patches so that

1. consistency between the ones with close boundaries is maximized, and

2. the visible surface of as many patches as possible is seen with their front-faces from as many viewpoints as possible.

Since we do not only consider solids (i.e., objects that have, or should have, a well-defined interior and exterior), it should be mentioned here, that for some models a consistent orientation of all normals is not possible (see Figure 2 for a simple example). In addition, with (intentionally) non-manifold models, it can become very hard to define the best of all possible orientations, even for humans. But even in those cases, our method will still find a good solution.

The results show that for almost all models our algorithm produces the desired normal orientation.

The next section will review some of the work related to ours. Section 3 will describe our algorithm, and after presenting some results in section 4, we will conclude in 5.

2. Related Work

So far, there are two approaches to solving the problem: *proximity-based* or *boundary-based*, and *solid-based*.

Proximity-based and boundary-based methods try to establish topological information based on the proximity of vertices or boundaries. In their surface reconstruction method, Hoppe et al. [6] determine a consistent orientation of tangent planes in all data points by solving a graph optimization problem. However, their method can be applied only to 2-manifold models. Other methods are inherently two-dimensional [7, 8].

Solid-based approaches try to partition $\mathbb{R}^3$ into cells that are either inside or outside the model (or the intended model) [14, 13]. Murali and Funkhouser [9] significantly extend this in order to deal with gaps, T-junctions, and intersections. However, these methods can handle only geometry that is closed and manifold (or intended to possess these properties).

Borodin et al. [2] close gaps in polygonal models by progressively connecting their boundaries. They identify close boundary edges in the same way, as we do in our method to determine boundary coherence. In [5], Guthe et al. collect the patch connectivity information in a so-called "seam graph", which is then used for view-dependent trimmed NURBS rendering.

In computational geometry, a lot of work has been devoted to robust computing [12, 3, 10]. However, these methods are not applicable here, because they try to avoid errors caused during the computation, while our algorithm tries to correct errors in the input data.

In this paper, we propose to combine the proximity-based approach with an approach we call *visibility-based*, thus fixing the problems of the proximity-based and circumventing those of the solid-based approaches.

3. Description of the Algorithm

In this section, we will first outline the algorithm, and then provide the details in the following subsections.

3.1. Outline

The input to our algorithm is an arbitrary polygon soup, i.e., a set of unorganized polygons without any explicit topology information, with or without normals. Since the orientation of each polygon's normal can be encoded in its vertex order, the problem of orientation of normals is equivalent to the problem of ordering vertices in polygons consistently. In the following, we will treat the term *polygon normal* synonymous with the term *vertex order*.

Usually, the output from modelling tools consists of a number of *patches*. Here, a patch consists of a set of polygons that are connected to each other, i.e., for which topology information can be constructed trivially. However, between patches there are usually more or less wide gaps. First, we build the neighbourhood information, divide the model into manifold surface patches, and detect their boundaries. At this point, we also orient the polygons consistently within each patch, which can be done trivially based on the topology information that is now available. After that, we determine those pairs of patches that are close to each other along some extent of their respective boundaries. For each such boundary pair we calculate its *coherence coefficient*.

Next, we determine the *visibility coefficients* for both sides of each patch. These coefficients describe how much of the surface of the patch is visible when viewed from all different viewing angles.

Finally, using both the boundary coherence and visibility coefficients we compute a global consistent orientation of the whole model. Patch boundaries that are close to each other make our algorithm consider normal orientation consistency more important than front-face visibility. On the other hand, patches that share only very loose boundaries with other patches are oriented such that front-face visibility is favored over normal consistency across the boundaries.

In the following, we will describe each step of the algorithm in detail.

3.2. Detection of Patches

As already mentioned, the input of our algorithm is a set of unorganized polygons. First, we read the input polygons and convert them to an indexed face set.

After that, we build the neighbourhood information for the mesh. In order to do so, we detect and collect all boundary and non-manifold edges. As boundary edges we define all edges which are incident to only one polygon. As non-manifold edges we define all edges which are incident to more than two polygons. During the traversal of the mesh we divide it into a set of manifold surface *patches*, which either are not connected with each other or connected only at vertices or non-manifold edges. For each patch we consistently orient all polygons belonging to it, which is trivial, due to the manifold topology that we now have. Of course, after that the orientation could differ between two neighbour patches, even if it was consistent in the original data.

The only problem that remains is how to orient the patches with respect to each other, which is not trivial, because their boundaries are only more or less close to each other along a part of that boundary.

3.3. Calculation of Boundary Coherence

At this stage we want to find close boundaries of different patches and determine the degree of their coherence. To accelerate finding pairs of close boundary edges we use a 3D grid, but many other spatial acceleration structures, such as k-d trees, can be used as well. Note that we should set a large search distance, as the quality of the results will suffer, if it is too small.

Assume that we have found the boundary edge e_j^n from patch P^n to be the closest neighbour for a boundary edge e_i^m from patch P^m. Then, we calculate the *local coherence*

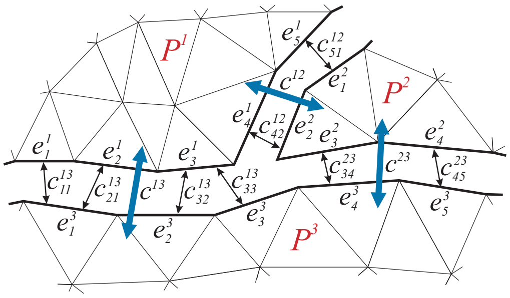


Figure 3. Local coherences c_{ij}^{mn} and coherence coefficients c^{mn} for patches P^1, P^2, P^3.

between these two edges as

$$c_{ij}^{mn} = -\operatorname{sgn}(s_{ij}^{mn}) \frac{\sqrt{\|s_{ij}^{mn}\|}}{1 + d_{ij}^{mn}}, \tag{1}$$

where $s_{ij}^{mn} = \vec{e}_i^m \cdot \vec{e}_j^n$ is the scalar product of $\vec{e}_i^m$ and $\vec{e}_j^n$, d_{ij}^{mn} is the shortest distance between e_i^m and e_j^n.[2]

The absolute value of the local coherence is approximately proportional to the edges' lengths and inversely proportional to the distance between them. Its sign shows whether the polygons incident to these boundary edges have the same or different orientation.

All local coherences for edge pairs from the patches P^m and P^n are summed up into the *coherence coefficient*:

$$c^{mn} = \sum_{i,j} c_{ij}^{mn}. \tag{2}$$

Figure 3 shows an example of the coherence coefficients. The idea of the boundary coherence coefficient is that it can give a hint as to which patches should probably be oriented consistently. This is, of course, only an intrinsic constraint.

3.4. Calculation of Visibility

We still need an external indicator to help choose the correct overall orientation of all patches. As mentioned before, our goal is to find a global orientation of patches, such that as many polygons as possible can be seen with their front-faces from most viewpoints. For this purpose, we want to determine the visibility of each side of each patch when seen from all possible viewpoints from outside the whole object. To do this, we can use various methods.

3.4.1. Ray shooting method. The first method we have tried is similar to a common raytracer. On the surface of each patch P^m we randomly and uniformly choose n^m points, where n^m is proportional to the area of P^m. Starting from each of these points we shoot a ray in a random direction. If the ray does not intersect any other polygons, we

[2] Note that all lengths are normalized by dividing them by the length of the longest side of the model's bounding box.

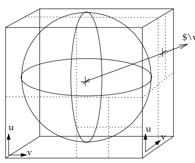


Figure 4. Ray space can be represented by a 5D cartesian rectangle with the help of the direction cube.

increment the counter for the polygon's side corresponding to the half-space in which the ray was shot. So, each patch has two counters n_f^m and n_b^m, which are the accumulation of all polygon counters. Finally, we define the *front-* and *back-face visibility coefficients* for each patch as

$$v_f^m = \frac{n_f^m}{n^m}, \quad v_b^m = \frac{n_b^m}{n^m}. \tag{3}$$

The main drawback of this method is that one needs to shoot a very large amount of rays in order to obtain an acceptable reliability.

3.4.2. 5D octree method. In the previous method we walk along a ray every time we shoot it into the scene. Of course, we accelerate this by any of the well-known data structures, such as k-d trees. However, rays are essentially static objects, just like the geometry of the model. So, based on the ideas of [1], we develop the following algorithm.

We discretize directions by the so-called *direction cube* (see Figure 4). Now we can build six octrees over the space of all rays, which are the 5-dimensional rectangles $R = U \times [-1, +1]^2 \times \{+x, -x, +y, -y, +z, -z\}$, where U is a suitable 3-dimensional bounding box around the complete model. Each cell, of the 5D octree corresponds to a beam in 3D geometric space, emanating from a 3D cell in geometric space [15].

Initially, we start with the root of the octree that comprises all possible rays. We associate all patches of our model with the root. Then, we recursively partition a node of the octree and distribute the set of patches among its children. A patch is associated with a child if it intersects the beam that child corresponds to. We stop the partitioning (i.e., conceptually we create a leaf), if either of the following conditions holds:

1. There is only one patch left in the 3D cell of the node. Now we must consider two sub-cases (see Figure 5):

   (a) There is no other patch associated with the node, i.e., all rays starting from the patch in the cube and in the direction of the beam would not hit any other patch (except for self-occlusions). Therefore, we add an amount to the visibility coefficient that corresponds to the spatial angle of the beam. Note that this angle depends only on the depth of the node, so the increase of the visibility coefficient can be precomputed.

   (b) There are other patches associated with the node. This means that at least some rays shot from within the 3D cell would hit another patch. Since we are only interested in approximations of the visibility coefficient, we assume that all rays would hit, and therefore we don't increase the coefficient.

2. The node's cell is too small (i.e., we have reached the maximum depth). Now we just consider the two sub-cases of the previous case for each of the patches that are (partially) inside the 3D cell of the node.

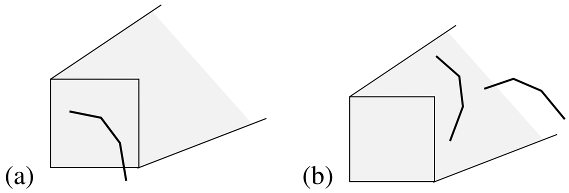


Figure 5. The two basic cases that can occur during computation of the visibility coefficients using a 5D octree over ray space.

Note again, that this computes, just like the previous of the following method, an approximation of the visibility coefficient. In our experience, though, such approximations are sufficient for all models we have tried so far.

Since we have now defined what a 5D cell of the octree represents, it is almost trivial to define how objects are assigned to sub-cells: we just compare the bounding volume of each object against the sub-cells 3D beam. Note that an object can be assigned to several sub-cells (just like in regular 3D octrees). The test whether or not an object intersects a beam could be simplified further by enclosing a beam with a cone, and then checking the objects bounding sphere against that cone. This just increases the number of false positives a little bit.

Note that the octree does not have to be built at all. As soon as we arrive at a leaf, we possibly increase the visibility coefficient, and then backtrack the recursion — we never actually construct any nodes. This greatly increases processing time. Furthermore, in contrast to [1], we need only very little memory, because we only keep some arrays of pointers to patches, one per recursion level. The number of levels is fairly limited (10–20).

3.4.3. GPU-based method. This method uses the GPU to calculate the visibility of single patches. The whole mesh is rendered from different points of view with colour coding, then the frame buffer is read and processed.

In order to get correct results we have to distribute the viewpoints uniformly around the model. We achieve this by placing them on the vertices of a tessellation of the bounding sphere of the model, produced by a successive subdivision of the icosahedron.

For each viewpoint we render the whole model onto a square viewport with a side length l_{vp}, using orthogonal projection, without shading and without anti-aliasing. Each side of each patch is drawn in an unique colour, which allows to unambiguously identify it when reading the pixels from the frame buffer. For each non-black pixel we increase the appropriate counter n_f^m or n_b^m (for front-face or back-face, respectively) of the patch P^m.

The side length l_{vp} should be chosen such that the smallest patch in the model will still occupy at least a few pixels from several viewpoints. Therefore, this method is only suitable for models where the ratio of the bounding box to the size of the smallest patch is not to large.

After n_t viewpoints, we define the front- and back-face visibility coefficients for each patch P^m as

$$v_f^m = \frac{n_f^m}{n_t \times l_{vp}^2 \times a^m}, \quad v_b^m = \frac{n_b^m}{n_t \times l_{vp}^2 \times a^m}, \tag{4}$$

where a^m is the area of patch P^m.

3.5. Consistent Orientation of Patches

After we have computed boundary coherence and visibility coefficients, we combine this information to find a consistent, global orientation of all surface patches.

For each patch P^m we already have its area a^m and two visibility coefficients v_f^m and v_b^m. We also have the set $\mathcal{C}$ of boundary coherence coefficients c^{mn}.

For each possible joint orientation of patches we define the overall front-face visibility V_f, back face visibility V_b, and coherence C of the super-patch as

$$\begin{aligned} V_f &= \frac{\sum(v_f^m \cdot a^m)}{\sum a^m}, \quad V_b = \frac{\sum(v_b^m \cdot a^m)}{\sum a^m}, \\ C &= \sum c^{mn}. \end{aligned} \tag{5}$$

Our goal is to find the orientation of all patches that maximizes both overall front-face visibility V_f and overall coherence C for all super-patches.

We will now repeatedly replace the set of patches $\mathcal{P}$ by a new set $\mathcal{P}'$, where two former patches $P^k, P^l \in \mathcal{P}$ have been joined conceptually into a *super-patch* $P^j \in \mathcal{P}'$. During this join operation, the orientation of one or both patches can be flipped.

In the following we will denote by the word *patch* either an original patch or a number of patches joined into one super-patch.

When we flip the orientation of a patch P^k, we update the coherence and visibility coefficients related to this patch in the following way:

$$\begin{aligned} \hat{v}_f^k = v_b^k, \quad \hat{v}_b^k = v_f^k, \\ \forall c^{mk} : \hat{c}^{mk} = -c^{mk}, \quad \forall c^{kn} : \hat{c}^{kn} = -c^{kn} \end{aligned} \tag{6}$$

In order to achieve a fast algorithm, we use a greedy strategy: in a queue, we sort all pairs of patches for which the boundary coherence is defined, so that the absolute values of the coherence coefficients c^{mn} are sorted in descending order. Then, we connect pairs of patches into super-patches, which get their own visibility coefficients. We also compute new boundary coherence coefficients between the new super-patch and the other patches and insert them into the queue.

More specifically, with each step we take a pair of patches P^m and P^n with the largest absolute coherence coefficient c^{mn} out of the queue. Their visibility coefficients are v_f^m, v_b^m, v_f^n, and v_b^n, respectively. Depending on a all these coefficients, we make a decision considering joint orientation of both patches.

3.5.1. Conforming coefficients.

If

$$\begin{array}{lllll} (c^{mn} > 0 & \wedge & v_f^m \geq v_b^m & \wedge & v_f^n \geq v_b^n) \quad \vee \\ (c^{mn} > 0 & \wedge & v_f^m \leq v_b^m & \wedge & v_f^n \leq v_b^n) \quad \vee \\ (c^{mn} < 0 & \wedge & v_f^m \geq v_b^m & \wedge & v_f^n \leq v_b^n) \quad \vee \\ (c^{mn} < 0 & \wedge & v_f^m \leq v_b^m & \wedge & v_f^n \geq v_b^n), \end{array}$$

then the visibility coefficients agree with the coherence coefficients. Therefore, we connect both patches into one super-patch S and define its front-face and back-face visibility coefficients as

$$\begin{aligned} v_f &= \frac{v_{max}^m \cdot a^m + v_{max}^n \cdot a^n}{a^m + a^n}, \\ v_b &= \frac{v_{min}^m \cdot a^m + v_{min}^n \cdot a^n}{a^m + a^n}, \end{aligned} \tag{7}$$

where $v_{max}^m = \max(v_f^m, v_b^m)$, $v_{min}^m = \min(v_f^m, v_b^m)$, a^m and a^n are the areas of the patches P^m and P^n. We also change orientation of one or both patches, if necessary (if $v_f < v_b$). If c^{mn} was negative, it becomes positive after the change of orientation (according to equations 6.

This choice of orientations results in the maximization of the front-face visibility v_f of the super-patch S and, at the same time, its consistent orientation.

3.5.2. Conflicting coefficients.

If

$$\begin{array}{lllll} (c^{mn} > 0 & \wedge & v_f^m \geq v_b^m & \wedge & v_f^n \leq v_b^n) \quad \vee \\ (c^{mn} > 0 & \wedge & v_f^m \leq v_b^m & \wedge & v_f^n \geq v_b^n) \quad \vee \\ (c^{mn} < 0 & \wedge & v_f^m \geq v_b^m & \wedge & v_f^n \geq v_b^n) \quad \vee \\ (c^{mn} < 0 & \wedge & v_f^m \leq v_b^m & \wedge & v_f^n \leq v_b^n), \end{array}$$

then the visibility coefficients come into conflict with the coherence coefficients: if we choose the patch orientations according to c^{mn}, the front-face visibility of the resulting super-patch will be not the maximum possible; on the other hand, the choice of orientations, which maximizes the front-face visibility, will result in inconsistent orientation on the boundary between the patches.

To find a tradeoff between the front-face visibility and boundary coherence, we compare them with some predefined values, which are parameters of the algorithm.

As already mentioned in 3.1, in case of close patch boundaries we consider normal orientation consistency more important than front-face visibility. Therefore, we first compare the boundary coherence with a threshold C_0. If $|c^{mn}/l^{mn}| > C_0$, where l^{mn} is the sum of lengths of all edges that contribute to c^{mn}, we assume the coherence between two patches to be strong and preserve their consistent orientation by connecting both patches into one super-patch. The visibility coefficients of the new super-patch are defined as

$$
\begin{aligned}
v_f = \max(v_1, v_2), \quad v_b = \min(v_1, v_2), \quad \text{where} \\
v_1 = \frac{v^m_{max} \cdot a^m + v^n_{min} \cdot a^n}{a^m + a^n}, \\
v_2 = \frac{v^m_{min} \cdot a^m + v^n_{max} \cdot a^n}{a^m + a^n}.
\end{aligned} \tag{8}
$$

If necessary, we change the orientation of one or both patches.

Otherwise, we compare the visibility coefficients off the two patches. If for one of the patches both visibility coefficients are very small or differ not much, and the visibility of the other patch dominates over it with respect to the patches' areas, we assume that its incorrect orientation will have only tiny impact on the overall front-face visibility. Therefore, if for the patch P^m

$$
\begin{aligned}
\left(v^m_b > \epsilon_v \ \wedge \ \frac{v^m_f}{v^m_b} < k_v \ \vee v^m_b < \epsilon_v \ \wedge \ v^m_f < \epsilon_v \right) \wedge \\
v_1 < v_2,
\end{aligned} \tag{9}
$$

holds, then we connect both patches into one super-patch. Here, ϵ_v is a lower threshold of visibility, below which we assume it is not important; k_v is a minimum ratio of largest to smallest visibilities of a patch, below which we assume their incorrect orientation is not important; v_1 and v_2 are defined in equation 8. The visibility coefficients of the new super-patch are calculated according to equation 8. If necessary, we change the orientation of one or both patches.

If condition 9 does not hold, we perform these comparisons again for the second patch P^n, and, if true, we still connect the two patches.

In all other cases we decide to favor front-face visibility over normal consistency across the boundaries and do not connect the patches. All their coefficients remain unchanged and we proceed to the next pair of patches from the priority queue.

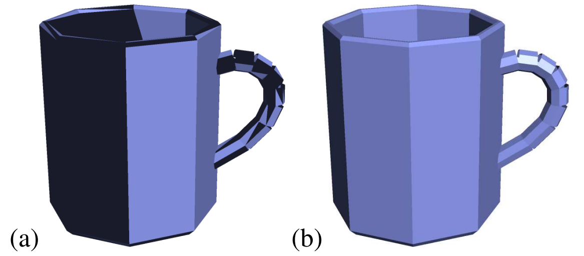


Figure 6. The coffee mug model used in [9]: (a) polygons are oriented randomly (back-facing polygons are drawn in black); (b) after applying our algorithm all polygons are oriented correctly

After the whole queue is processed, we get the final global orientation of all surface patches.

4. Results

We have tested our method with a large number of models consisting of up to 3,000 surface patches.

Our new algorithm produced the desired orientation for almost all models of our test suite. Figure 7 shows a sample of our test suite. Our method can also handle the model shown in Figure 6, which Murali and Funkhouser reported to be difficult for *proximity-based* approaches.

Additionally, we tested the robustness of our algorithm. To this end, we changed the orientation of a random sample of the input polygons (see Figure 6a). This had no influence on the result, i.e., our algorithm does not depend on the initial orientation of the input and all normals are correct afterwards.

We have also investigated whether only one of the two criteria (boundary coherence or visibility) would be sufficient. Our tests have shown that for many models the independent maximization of front-face visibility for each single patch can be sufficient. Using boundary coherence only was successful with some models. However, there are models where both criteria are needed. Figures 8 and 9 show examples where using only one of the two criteria fails to produce the desired result.

Table 1 shows the performance rates of our algorithm for 3 different models, each at three different levels of detail. To calculate the visibility coefficients we used the GPU-based method with 80 viewpoints and viewport of 400x400 pixels. The timings have been obtained on Pentium 4/1.8GHz with GeForce2 MX. Obviously, the overall times are dominated by the visibility computation. This time and also the time for building the topology information and detecting the patches are mostly linear in the number of polygons. The time for calculating the boundary coherence coefficients depends mostly on the number of boundary edges. Apparently, for the first model the simplification tool performed almost no reduction on boundaries.

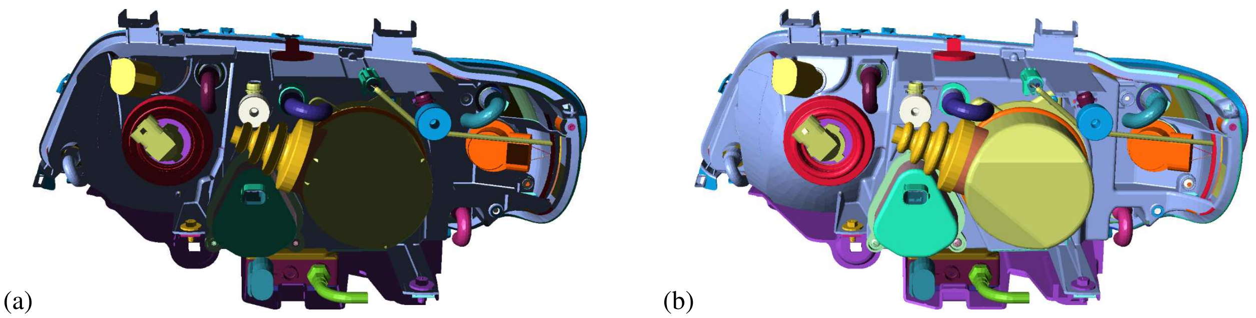


Figure 7. A model consisting of 323 patches, which are shown in different colours. Dark lighting shows back-facing polygons, which denote incorrect orientation. (a) Original model; (b) after applying our algorithm, all patches are correctly oriented.

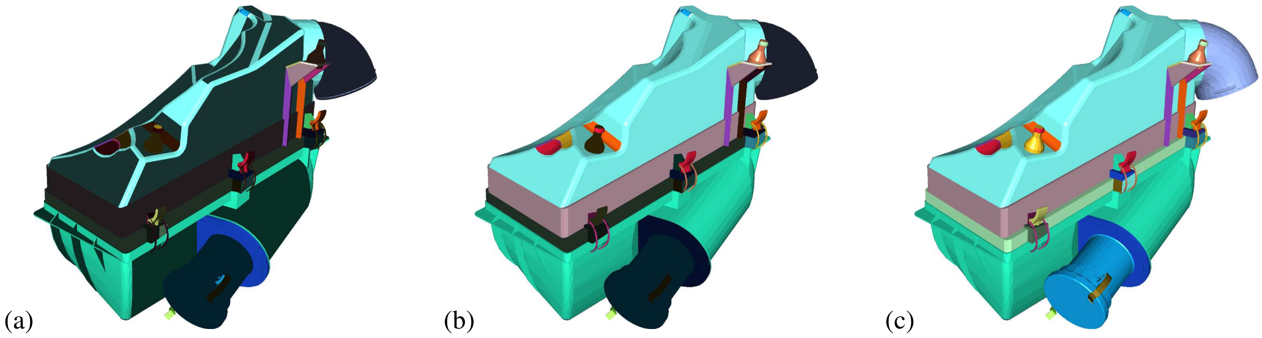


Figure 8. A model with 71 patches: (a) original model; (b) after applying our algorithm with only boundary coherence taken into account; (c) taking both boundary coherence and visibility into account.

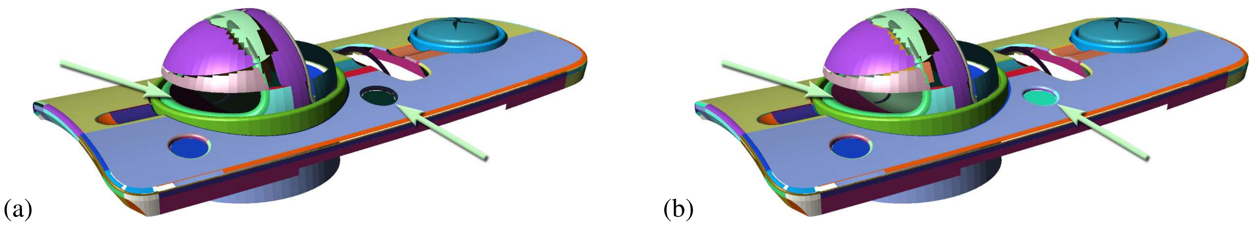


Figure 9. A model with 137 patches: (a) after applying our algorithm with only visibility taken into account; some incorrectly oriented patches are marked with arrows; (b) taking both boundary coherence and visibility into account.

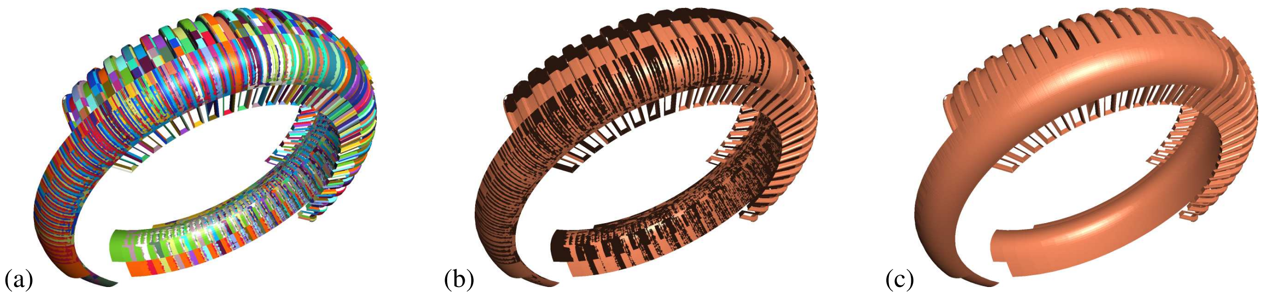


Figure 10. A model with 1250 patches: (a) original model with patches shown in different colours and lighted from both sides – many patches overlap or are coplanar with each other; (b) after applying our algorithm – several patches are oriented incorrectly; (c) same model lighted from both sides.

	No. of polygons	No. of patches	Patch time (s)	Coher. time (s)	Visib. time (s)
1	180 252	78	5.6	0.7	17.7
	60 080	80	1.5	0.6	6.6
	18 019	83	0.4	0.5	2.5
2	194 668	1 508	5.9	3.4	19.0
	64 648	1 509	1.8	2.0	6.8
	19 142	1 511	0.4	1.1	2.6
3	300 836	3 310	10.5	9.2	29.1
	84 673	2 040	2.7	2.8	9.1
	14 327	2 067	0.4	0.9	2.5

Table 1. Performance rates of our algorithm for 3 models at different levels of detail: time for detecting the patches, time for calculating the boundary coherence coefficients, and time for calculating the visibility coefficients.

It is difficult to compare the performance of our algorithm with that of Murali and Funkhouser without re-implementing their approach. Therefore, we tried to convert the timings reported by them on our platform. We used a scaling factor of 50, which means that their algorithm could handle a model of about 1200–1600 polygons in about 1.5–4.5 seconds. In contrast, our method can handle a model of 15000–18000 polygons in 3.4–4.1 seconds.

5. Conclusions and Future Work

We have proposed a new approach, which we call *visibility-based*, to the problem of consistently and sensibly orienting all normals of arbitrary polygonal models. We combine this approach with a *proximity-based* approach, which yields a method that can correctly orient more models than previous methods.

Our method produces the desirable solution for almost all practical cases. Only in cases with a lot of coplanar polygons it produces sub-optimal results.

The algorithm is controlled by only very few parameters, and their adjustment is not critical. While some of them balance the tradeoff between accuracy and speed, the others determine the choice between consistent orientation and visibility.

As explained earlier, our method is applicable to objects consisting of other primitives as well, such as NURBS. This is one avenue for future work.

Another area is the search of other metrics and other criteria for decision-making in conflict situations. Also, acceleration and increase of accuracy of our method could be considered as further development.

Acknowledgements

We would like to thank Thomas Funkhouser for providing us with the coffee mug model and our colleague Michael Guthe for his valuable advice. The bot model is courtesy of Michael Beals. Models of automotive parts are courtesy of DaimlerChrysler AG.

References

[1] J. Arvo and D. Kirk. Fast ray tracing by ray classification. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):55–64, July 1987.

[2] P. Borodin, M. Novotni, and R. Klein. Progressive gap closing for mesh repairing. In J. Vince and R. Earnshaw, editors, *Advances in Modelling, Animation and Rendering*, pages 201–213. Springer Verlag, July 2002.

[3] S. Fortune and C. J. Van Wyk. Efficient exact arithmetic for computational geometry. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 163–172, May 1993.

[4] M. Garland and P. S. Heckbert. Simplifying surfaces with color and texture using quadric error metrics. In D. Ebert, H. Hagen, and H. Rushmeier, editors, *IEEE Visualization '98*, pages 263–270, 1998.

[5] M. Guthe, J. Meseth, and R. Klein. Fast and memory efficient view-dependent trimmed nurbs rendering. In S. Coquillart, H.-Y. Shum, and S.-M. Hu, editors, *Pacific Graphics 2002*, pages 204–213. IEEE Computer Society, October 2002.

[6] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2):71–78, 1992.

[7] B. W. Kernighan and C. J. van Wyk. Extracting geometrical information from architectural drawings. In *Proc. of the Workshop on Applied Computational Geometry*, pages 82–87, May 1996.

[8] R. Laurini and F. Milleret-Raffort. Topological reorganization of inconsistent geographical databases: a step towards their certification. *Computer and Graphics*, 18(6):803–813, 1994.

[9] T. M. Murali and T. A. Funkhouser. Consistent solid and boundary representations from arbitrary polygonal data. In *Symposium on Interactive 3D Graphics*, pages 155–162, 196, 1997.

[10] M. Segal. Using tolerances to guarantee valid polyhedral modeling results. In F. Baskett, editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 105–114, Aug. 1990.

[11] X. Sheng and I. R. Meier. Generating topological structures for surface models. *IEEE Computer Graphics and Applications*, 15(6):35–41, 1997.

[12] J. R. Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete Comput. Geom.*, 18(3):305–363, 1997.

[13] S. Teller and P. Hanrahan. Global visibility algorithms for illumination computations. In *Computer Graphics Proceedings, Annual Conference Series, 1993*, pages 239–246, 1993.

[14] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. In M. C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 153–162, July 1987.

[15] G. Zachmann and E. Langetepe. Geometric data structures for computer graphics. In *SIGGRAPH '03 Proceedings*. ACM Transactions of Graphics, July 2003. Tutorial.