

A Simple Method for Correcting Facet Orientations in Polygon Meshes Based on Ray Casting

Kenshi Takayama

Department of Computer Science
ETH Zurich

Alec Jacobson

Department of Computer Science
Columbia University

Ladislav Kavan

Department of Computer and Information Science
University of Pennsylvania

Olga Sorkine-Hornung

Department of Computer Science
ETH Zurich



Figure 1. A few examples of meshes from the SHREC'10 Generic 3D Warehouse dataset [Vanamali et al. 2010] with incorrect facet orientations (top row), and the results of our automatic correction method (bottom row).

Abstract

We present a method for fixing incorrect orientations of facets in an input polygon mesh, a problem often seen in popular 3D model repositories, such that the front side of facets is visible from viewpoints outside of a solid shape represented or implied by the mesh. As opposed to previously proposed methods which are rather complex and hard to reproduce, our method is very simple, only requiring sampling visibilities by shooting many rays. We also propose a simple heuristic to handle interior facets that are invisible from exterior viewpoints. Our method is evaluated extensively with the SHREC'10 Generic 3D Warehouse dataset containing 3168 manually designed meshes, and is demonstrated to be very effective.

1. Introduction

Polygon meshes are a fundamental primitive in computer graphics and other fields that rely on geometric manipulation, from geoscience to medicine. A polygon mesh generally consists of a set of 3D vertex positions and a set of polygonal facets each defined by an ordered list of vertex indices. The ordering of vertices of a facet determines its orientation; i.e., by convention, if we see the ordering of vertices of a facet as counterclockwise from a particular viewpoint, we consider the facet to be showing its front side. Having “correct” facet orientations is essential for various applications such as back face culling, rendering with two-sided colors, inside-outside segmentation [Jacobson et al. 2013], and collision response [Harmon et al. 2008]. Here, the “correctness” of facet orientations is defined as follows: a facet’s orientation is correct if its front side corresponds to the “outside” of a solid object represented or implied by the mesh. This definition is not applicable to a mesh which does not represent or imply a solid object, a common example of which is a “single-faceted” geometry representing an infinitely thin part with only one sheet of facets, see Figure 7. Such cases are excluded from the scope of this paper.

Unfortunately, it is still common today to encounter polygon meshes with incorrect facet orientations when working from Internet repositories, including commercial sites such as TurboSquid. We speculate that one cause is that many popular polygon-based 3D modeling tools, such as SketchUp, use double-sided lighting that ignores facet orientations. This allows modelers to create polygon meshes with arbitrary facet orientations and provides no visual feedback on the error in the content creation tool. Not all tools have this problem. For example, solid modeling tools such as ZBrush and CATIA produce solid shapes with a clearly-defined inside and outside that export without error to surface meshes.

Incorrect facet orientations may also arise when compositing multiple meshes from different sources. The top row of Figure 1 shows a few examples of such incorrect facet orientations found in the popular SHREC’10 Generic 3D Warehouse dataset [Vanamali et al. 2010]. In this paper, we describe an automatic method for correcting such facet orientations. The bottom row of Figure 1 shows a sample result by this method.

1.1. Previous methods

There exist three methods in the literature for correcting facet orientations in polygon meshes. **Murali and Funkhouser’s method** [1997] computes a discrete harmonic function over a tessellation of the 3D space (either via Binary Space Partitioning or tetrahedralization) whose sign encodes whether a given 3D position is inside (positive) or outside (negative) of a solid shape implied by the input polygon mesh. Each facet can be reoriented by integrating the function values over its area on its front and back sides separately and comparing the results. Because the function values on the

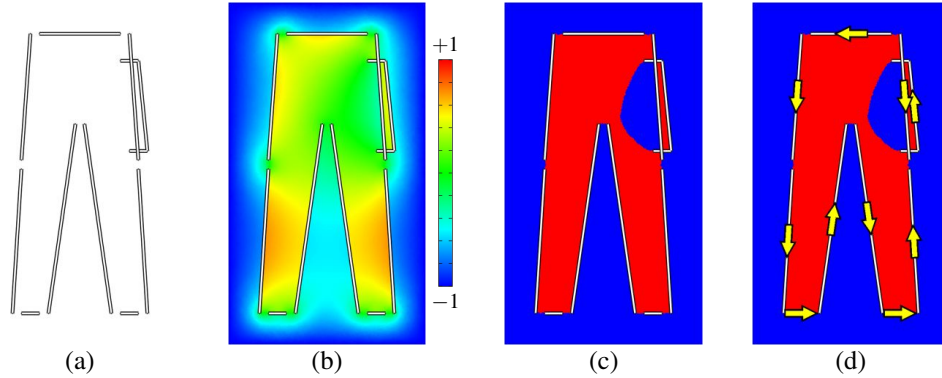


Figure 2. An example in 2D (representing a pants with an accessory) where Murali and Funkhouser’s method [1997] fails because of overlapping facets: (a) input facets, (b) visualization of the harmonic function, (c) binary coloring of the function, (d) result of orienting each facet based on the sum of the function values on its two sides.

two sides of every facet are constrained to have the same absolute value but opposite signs, the function value vanishes where the input mesh has open boundaries or intersections, leading to unconfident indication of insideness or outsideness. In addition, this method assumes that every facet lies on the boundary between inside and outside of the implied solid shape, but such an assumption does not hold for meshes that are not meant to imply a solid shape. Even if the mesh does represent a solid shape, the assumption is wrong if there are some overlapping facets, often leading to unintended facet orientations (Fig. 2). Additionally, robustly tessellating the 3D space while conforming to an arbitrary polygon mesh is far from trivial, which is another drawback of this method.

Borodin et al.’s method [2004] and **Zhou et al.’s method [2008]** both sample visibilities of facets by shooting many rays from outside the bounding box of the input mesh to get an “initial guess” of facet orientations, followed by subsequent steps of greedy aggregation (Borodin et al.) and graph-cut (Zhou et al.) to obtain final facet orientations. However, it is unclear to what degree these subsequent steps improve facet orientations computed in the visibility sampling step. In fact, Borodin et al. claim in their Figure 9 that visibility sampling alone is insufficient and problematic, which we question because visibility sampling should orient facets such that as few as possible facets show their back sides to exterior viewpoints. Reproducing and analyzing their results seem difficult, since quite a few hand-tuned parameters are involved in these methods, and their reference implementations and datasets are not readily available.

In this paper, we present a very simple method based solely on visibility sampling for correcting facets in polygon meshes. We evaluate our method with the SHREC’10 Generic 3D Warehouse dataset and show that our method works sufficiently well for

all the cases where the problem is well defined.

2. Method

2.1. Visibility sampling

Our method builds on Borodin et al.'s [2004] strategy for visibility sampling, which is described in Section 3.4.1 of their paper. The basic idea is to orient each facet such that its front side is more visible from outside, which is determined by shooting many rays from the facet. Specifically, for each facet, a large number of points proportional to the facet's area are randomly sampled on the facet as the ray origins, and for each ray origin, a ray direction is randomly sampled. Our modification to Borodin et al.'s method is that for each ray origin and direction, we shoot two rays in opposite directions to ensure that the same number of rays are shot for both sides of the facet. If a ray shot on the facet's front (resp. back) side does not intersect with any other facets in the mesh, the corresponding counter c_{front} (resp. c_{back}) is incremented. After shooting all the rays, the facet is flipped if $c_{\text{front}} < c_{\text{back}}$.

Handling interior facets. The above method cannot make any decisions for facets that are completely occluded and invisible from outside (often representing some internal structures, see Figure 8) because c_{front} and c_{back} are both 0. We propose a simple heuristic to handle such cases, where the basic idea is to orient an interior facet such that the front side has more “free” space than the back side. Specifically, in the ray shooting process as explained above, if a ray shot on the facet's front (resp. back) side intersects with some other facet, the distance between the ray origin and the intersecting point is accumulated to the corresponding counter d_{front} (resp. d_{back}). After shooting all the rays, the facet is flipped if

$$(c_{\text{front}} < c_{\text{back}}) \text{ or } ((c_{\text{front}} = c_{\text{back}}) \text{ and } (d_{\text{front}} < d_{\text{back}})). \quad (1)$$

2.2. Ray intersection parity sampling

If the input mesh implies a solid shape and it has a narrow cavity invisible from the outside, the above heuristic method may orient the facet at the cavity in a way that contradicts with the implied solid shape (Figure 3b). For such cases, although they seldom occur and are mostly of theoretical interest (see Section 3), we propose an alternative method based on *ray intersection parity* (Figure 3c), which is achieved with a slight modification to the above method. Specifically, the system employs the same ray shooting process as the above method, but this time, for each ray, it detects all the intersections with all the facets (i.e., without stopping at the first intersection), and accumulates the number of intersections *modulo 2* to counters p_{front} or p_{back} corresponding to the ray direction. After shooting all the rays, the facet is flipped if

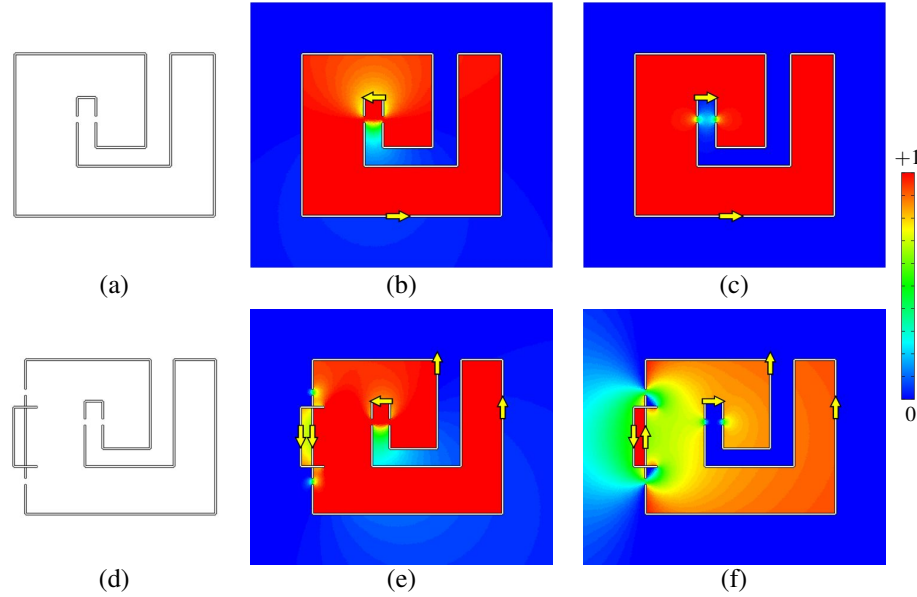


Figure 3. 2D examples demonstrating the effect of the alternative method based on ray intersection parity. (a, d) Input facets. (b, e) Result of the visibility sampling method with the heuristic to handle the interior facet. (c, f) Result of the alternative method based on ray intersection parity. The color map visualizes the generalized winding number function [Jacobson et al. 2013] defined by the given facet orientations.

$$P_{\text{front}} > P_{\text{back}}.$$

Because this method assumes that the input mesh implies a solid shape, it may produce some unintuitive facet orientations if the assumption is violated, similar to Murali and Funkhouser’s method [1997]. If the input mesh contains both overlapping facets and facets representing internal cavities invisible from outside, neither of the above two methods produce facet orientations that we would expect (Figure 3e-f). However, such cases were never observed in the dataset we used in the evaluation.

3. Evaluation

Dataset. We used the SHREC’10 Generic 3D Warehouse dataset [Vanamali et al. 2010] to evaluate the performance of our method. This dataset was created by first collecting SketchUp (*.skp) files from the Google 3D Warehouse and then manually processing them (e.g., decomposing a scene with many related objects into individual objects) so that the dataset is suitable for benchmarking shape retrieval algorithms. It contains 3168 meshes, most of which are far from being clean and watertight: they very often have various types of defects such as disconnected facets with random orientations and self-intersecting/coplanar facets. We believe this dataset to be a fairly good representation of meshes manually created by general users.

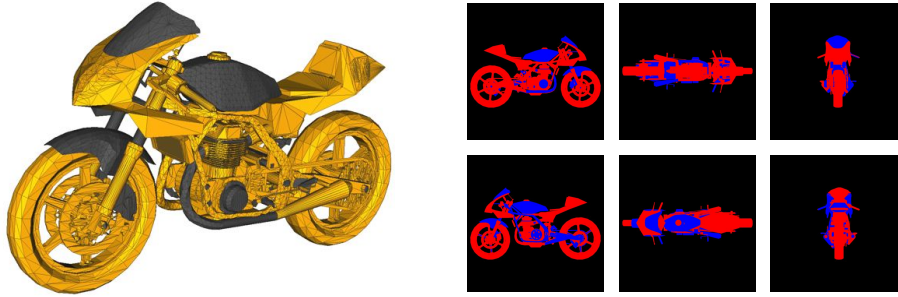


Figure 4. A motorcycle model (left, named `2292.off` in the dataset) whose *backfacingness* is measured to be 0.2756 according to rendered images with two-sided coloring (right).

Note that meshes in the dataset often contain single-faceted facets representing infinitely thin objects (e.g., fish fins) or facets invisible from outside representing interior objects (e.g., car seats). In such cases, the “correct” facet orientations can only be decided by the authors of the meshes according to their intentions, which makes it impossible to define the ground-truth facet orientations for the dataset. Therefore, in the following, we do not discuss the performance of our method in terms of “accuracy”.

Backfacingness measure. Nevertheless, we use a measure called *backfacingness* which tells us how much of a given mesh’s facets are showing their back sides to distant viewpoints, to measure the plausibility of the facet orientations at least when viewed from outside. To compute this measure, we first render the mesh into a framebuffer with two-sided coloring and orthographic projection from a few view directions (Fig. 4). The measure is then obtained as the ratio of the number of pixels coming from the back side to the total number of drawn pixels. We found that the framebuffer resolution of 1024^2 and 6 canonical view directions provide sufficiently good reliability in practice. Note again that this measure is irrelevant for single-faceted parts and interior objects/cavities where neither of the two orientations is more plausible than the other in general.

Preprocessing. We implemented a preprocessing step of extracting manifold patches similar to Borodin et al.’s method (in Section 3.2 of their paper) where the basic idea is to group facets connected by manifold edges (i.e., edges shared by exactly two facets) into a manifold patch. However, we found that there are quite a few cases where the extracted patch structures are rather unexpected because of the complicated mesh connectivities designed by hand. Figure 5a shows an example where the facets for the top cover and the bottom support of the piano model are grouped as patches in an undesirable way. To circumvent this problem, we omit the patch extraction step in a default setting and process individual facets independently (Fig. 5b). Note, how-

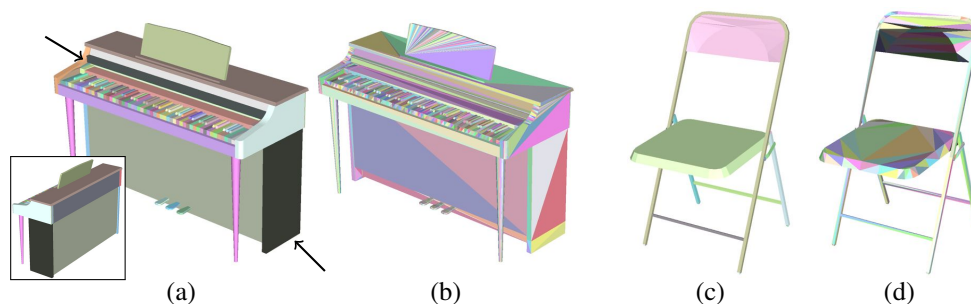


Figure 5. Pros and cons of patch-wise and facet-wise processing. Patch IDs are visualized with different colors, and backfacing facets are rendered darker. In some cases, manifold patches are extracted in an unexpected manner (a) because of the complicated mesh connectivities in the input meshes; in this case of the piano model (2371.off), there are non-manifold edges in the base of the piano, and decomposing them into oriented manifold patches led to an unexpected result with the top cover and the bottom support being grouped together, whereas processing each facet independently is more appropriate (b). However, for single-faceted geometries whose front-side and back-side visibilities are mostly the same, such as the back seat of the chair model (938.off), extracting manifold patches and processing patch-wise is more appropriate (c), otherwise the visibility-based decision for each facet can fluctuate (d).

ever, that this setting can be inappropriate for meshes that represent thin parts with single-faceted facets; for such cases, omitting the patch extraction step may result in some neighboring facets oriented in opposite directions (Fig. 5d). Another preprocessing step we employ as a default setting is to eliminate facets with identical vertex sets. Quite a few meshes in the dataset contain multiple facets that have the exact same set of vertices (in the same or reversed orientation). Such facets can lead to the generation of numerous spurious patches, and may also confuse the ray intersection engine.

Results. We applied our method with the default settings (i.e., patch extraction being skipped, all facets having unique vertex sets) to all the 3168 meshes in the dataset. The average and the standard deviation of the *backfacingness* measure are 0.19 and ± 0.26 for the original meshes, and 0.0026 and ± 0.0063 for the output meshes, respectively. Figures 1 and 6 show some results where the output meshes are oriented in a natural way. Figure 7 shows some output meshes with higher *backfacingness* remaining. Note that all the facets showing their back sides belong to single-faceted geometry representing very thin parts, and therefore there is no way to reduce the *backfacingness* any further. Figure 8 shows some results containing interior objects invisible from the outside, such as car seats. Note how our heuristic described in Section 2.1 works successfully for orienting interior objects. By manually and carefully inspecting each mesh in the database, we confirmed that there are no cases where our



Figure 6. Results of our method. In each pair, the original facet orientations and the facet orientations corrected by our method are shown on the left and on the right, respectively.

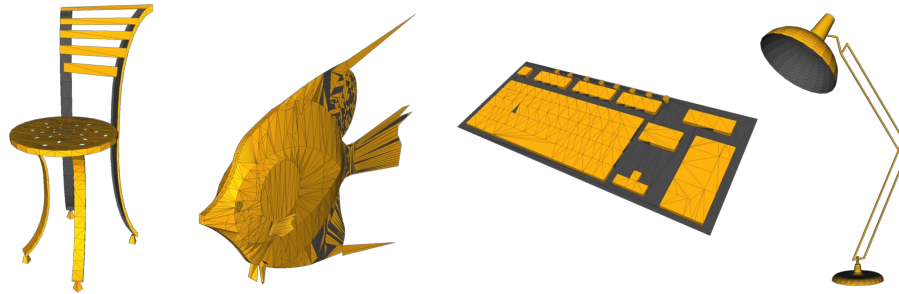


Figure 7. Examples where the *backfacingness* measures remain relatively high after processing, due to the single-faceted geometry.

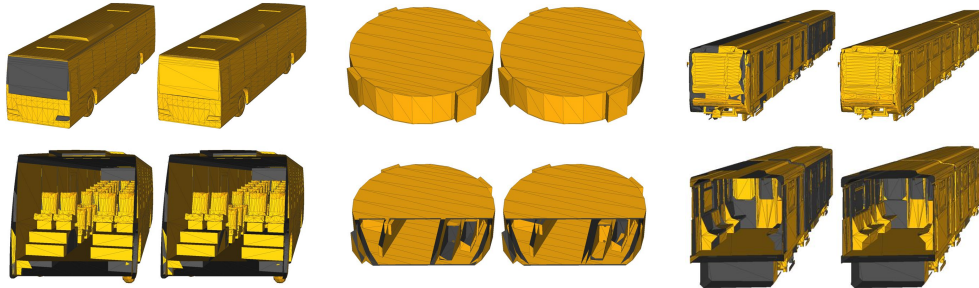


Figure 8. Results with meshes containing interior objects. In each set, the original and the processed facet orientations are shown on the left and right columns, respectively. The top and bottom rows show the exterior views and the cut-away views exposing the interior, respectively.

alternative method described in Section 2.2 is more suitable. Table 1 shows how much *backfacingness* was reduced for each of these results.

Fig	Model (ID)	Backfacingness	Fig	Model (ID)	Backfacingness
1	Bicycle (103)	0.3743 → 0.0085	6	Woman (3120)	0.5343 → 0.0003
	Cell phone (835)	0.3000 → 0.0287	7	Chair 3 (914)	0.5755 → 0.3743
	Chair 1 (994)	0.6344 → 0.0880		Fish (1335)	0.1499 → 0.1499
	Truck 1 (2983)	0.2458 → 0.0344		Keyboard (1635)	0.4819 → 0.4811
6	Bird (164)	0.5535 → 0.0073		Lamp (1953)	0.4803 → 0.2477
	Chair 2 (994)	0.6344 → 0.0880	8	Bus (619)	0.0544 → 0.0001
	Guitar (1437)	0.5556 → 0.0009		Parts (1065)	0.0000 → 0.0000
	Lamp 1 (1990)	0.5868 → 0.0273		Train (2800)	0.3002 → 0.0020
	Truck 2 (2867)	0.5693 → 0.0050			

Table 1. The reduction of *backfacingness* for each of the results in the paper. The ID of a model denotes its file name in the dataset (e.g., a model with ID 103 is named as 103.off).

Performance. We implemented our method in C++ using the libigl¹ library for geometry manipulation and the Embree² library for ray tracing. We include the source code of this implementation with our supplementary material. Processing the largest mesh in the dataset, 1938.off with 16k vertices and 92k triangles, using 3 million rays takes about 10 seconds under this implementation on an Intel Core i7 2.6 GHz CPU with 8 GB of RAM.

Acknowledgements

We are grateful to Peter Kaufmann, Wenzel Jakob, Nobuyuki Umetani, Melina Skouras, Ilya Baran, and Ryan Schmidt for their advice and feedback. This work was supported in part by the ERC grant iModel (StG-2012-306877), by an SNF award 200021_137879, by a NSF grant IIS-1350330, and by a gift from Adobe Research. Kenshi Takayama’s work was funded in part by JSPS Postdoctoral Fellowships for Research Abroad. Alec Jacobson’s work was supported in part by an Intel Doctoral Fellowship.

References

- BORODIN, P., ZACHMANN, G., AND KLEIN, R. 2004. Consistent normal orientation for polygonal meshes. In *Proceedings of the Computer Graphics International, CGI ’04*, 18–25. URL: <http://dx.doi.org/10.1109/CGI.2004.32>. 55, 56
- HARMON, D., VOUGA, E., TAMSTORF, R., AND GRINSPUN, E. 2008. Robust treatment of simultaneous collisions. *ACM Transactions on Graphics (Proc. of SIGGRAPH)* 27, 3, 23:1–23:4. URL: <http://doi.acm.org/10.1145/1360612.1360622>. 54
- JACOBSON, A., KAVAN, L., AND SORKINE-HORNUNG, O. 2013. Robust inside-outside segmentation using generalized winding numbers. *ACM Transactions on Graphics*

¹<http://igl.ethz.ch/projects/libigl/>

²<http://embree.github.io/>

(*Proc. of SIGGRAPH*) 32, 4, 33:1–33:12. URL: <http://doi.acm.org/10.1145/2461912.2461916>. 54, 57

MURALI, T. M., AND FUNKHOUSER, T. A. 1997. Consistent solid and boundary representations from arbitrary polygonal data. In *Proceedings of the 1997 Symposium on Interactive 3D Graphics, I3D '97*, 155–163. URL: <http://doi.acm.org/10.1145/2533284.2533286>. 54, 55, 57

VANAMALI, T. P., GODIL, A., DUTAGACI, H., FURUYA, T., LIAN, Z., AND OHBUCHI, R. 2010. Shrec'10 track: Generic 3D warehouse. In *Proceedings of the 3rd Eurographics Conference on 3D Object Retrieval, 3DOR '10*, 93–100. URL: <http://dx.doi.org/10.2312/3DOR/3DOR10/093-100>. 53, 54, 57

ZHOU, K., ZHANG, E., BITTNER, J., AND WONKA, P. 2008. Visibility-driven mesh analysis and visualization through graph cuts. *IEEE Transactions on Visualization and Computer Graphics* 14, 6, 1667–1674. URL: <http://dx.doi.org/10.1109/TVCG.2008.176>. 55

Index of Supplemental Materials

- `code/`: C++ implementation of our algorithm configured with CMake.
- `data/`: Meshes whose facet orientations have been corrected by our algorithm, and reports of *backfacingness* for all input and output meshes considered in our experiments.
- `py/`: Some of small Python scripts to batch-process meshes.

Author Contact Information

Kenshi Takayama
Department of Computer Science
ETH Zurich
CAB G 88, Universitaetstrasse 6
8092 Zurich, Switzerland
kenshi84@gmail.com

Alec Jacobson
Department of Computer Science
Columbia University
Mail Code 0401
500 West 120 St, Room 450
New York, NY 10027
jacobson@cs.columbia.edu

Ladislav Kavan
Department of Computer and
Information Science
University of Pennsylvania
Levine Hall 303, 3330 Walnut Street
Philadelphia, PA 19104-3409
ladislav.kavan@gmail.com

Olga Sorkine-Hornung
Department of Computer Science
ETH Zurich
CNB G 106.2, Universitaetstrasse 6
8092 Zurich, Switzerland
sorkine@inf.ethz.ch

Kenshi Takayama and Alec Jacobson and Ladislav Kavan and Olga Sorkine-Hornung, Correcting Facet Orientations in Polygon Meshes, *Journal of Computer Graphics Techniques (JCGT)*, vol. 3, no. 3, 53–63, 2014
<http://jcgt.org/published/0003/04/02/>

Received: 2014-06-18

Recommended: 2014-08-16

Published: 2014-11-21

Corresponding Editor: Bernd Bickel

Editor-in-Chief: Morgan McGuire

© 2014 Kenshi Takayama and Alec Jacobson and Ladislav Kavan and Olga Sorkine-Hornung (the Authors).

The Authors provide this document (the Work) under the Creative Commons CC BY-ND 3.0 license available online at <http://creativecommons.org/licenses/by-nd/3.0/>. The Authors further grant permission reuse of images and text from the first page of the Work, provided that the reuse is for the purpose of promoting and/or summarizing the Work in scholarly venues and that any reuse is accompanied by a scientific citation to the Work.

