Removing self-intersections in 3D meshes while preserving floating-point coordinates

Sylvain Lazard, Leo Valque

Removing self-intersections in 3D meshes while preserving floating-point coordinates

Sylvain Lazard* Léo Valque*

January 22, 2025

Abstract

We present a straightforward and robust method for resolving the mesh intersection problem. We focus specifically on the challenge caused by the intersections resulting from the conversion of the vertices coordinates from their exact mathematical values to a fixed-precision floating-point format. Our method outputs intersection-free models whose vertices coordinates are all represented with double-precision floating-point format. We evaluated our approach thoroughly, considering a large collection of meshes. In particular, we can process all the 4 524 models in Thingi10K [Zhou and Jacobson, 2016] that contain self-intersections. This outperforms previous state-of-the-art approaches: On the 527 models of Thingi10K for which naive rounding fails, Zhou et al. [2016]'s approach is capable of handling 91% of them, and Valque [2024]'s 94%. In terms of time efficiency, our approach handles about 17k vertices per second (serial) on average, which is faster to that of Zhou et al. [2016] by a factor 1.7 on these non-trivial models and is faster than that of Valque [2024] by several order of magnitude.

1 Introduction

In computer science, many software pipelines rely on clean, well-structured 3D meshes and, in particular, meshes without self-intersections. However, 3D meshes are very often not clean: for instance, about half of the 10 000 "real-world" meshes from the popular Thingi10K repository [Zhou and Jacobson, 2016] contain self-intersections. A standard and natural approach to solving that issue is to compute the mathematically exact self-intersections and subdivide the faces accordingly. However, most software dealing with polygonal objects requires input vertices whose coordinates have single- or double-precision floating-point format (referred to as float or double). The coordinates of the vertices resulting from the intersections thus need to be rounded from their exact mathematical values to fixed-precision floating-point representations. We refer to this approach as naive rounding.

Surprisingly, naive rounding often fails in the sense that the rounding of the coordinates induces new self-intersections between the faces. For instance, out of the 4 524 models in Thingi10K that contain self-intersections, naive rounding fails on 527 models, that is in about 12% of these models. It should be stressed that, in theory, it is unknown whether iterating the naive rounding scheme always terminates and, in practice, iterating for at most one hour fails fairly often, namely, on about half (273) of these 527 models.

When addressing the long-lasting challenge of robustly handling mesh intersections, a common weakness in all robust methods that compute mathematically exact intersection vertices is the

*Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France, `Firstname.Name@loria.fr`

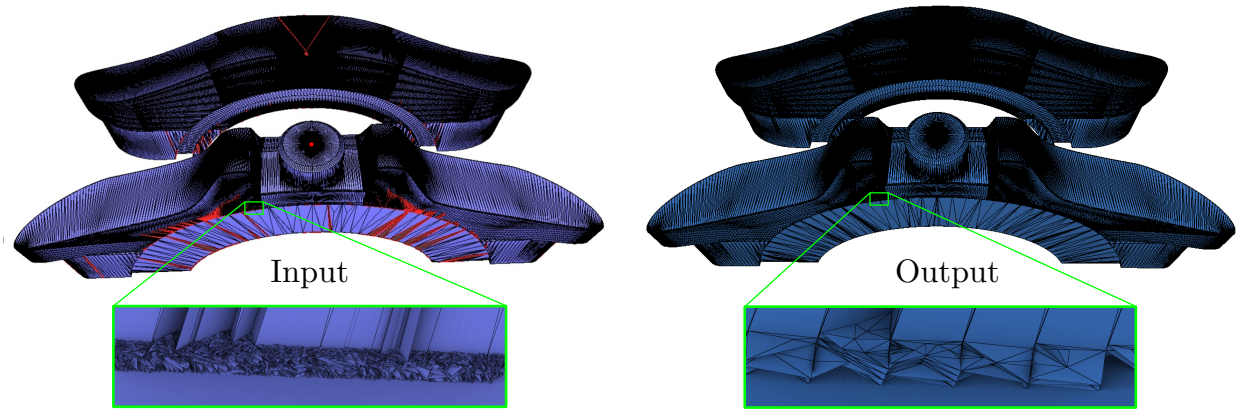


Figure 1: The famously difficult model of Thingi10K [Zhou and Jacobson, 2016] referenced as number 996 816, which contains about 76k vertices, 171k triangles, 25 millions pairs and triplets of intersecting triangles (shown in red), and whose mathematically exact intersection-free representation contains about 30 millions vertices and 240 millions faces. We resolve the intersections in 38 seconds (serial) on this model. The output model contains about 100k vertices and 245k faces, with all coordinates in doubles with a relative error of at most 6.10^{-5}.

conversion, or rounding, of their exact coordinates to fixed-precision floating-point format; see e.g. [Zhou et al., 2016, Cherchi et al., 2020, Livesu, 2024, Lévy, 2024, Cherchi et al., 2022, Attene, 2020, Diazzi and Attene, 2021, Trettner et al., 2022]. As mentioned by Cherchi et al. [2022]: "Solving the snap rounding problem is the ultimate solution for all these issues, but this is a remarkably difficult problem".

To the best of our knowledge, only two (non-trivial) practical approaches have been proposed for handling this chalenge. Zhou et al. [2016] proposed a simple heuristic which is capable of handling 91% of the 527 *non-trivial* models of Thingi10K, i.e., those for which the naive rounding fails. Very recently, Valque [2024] proposed a practical algorithm, which handles 94% of the non-trivial models. This algorithm also provides some topological guarantees but its implementation is difficult and it is several order of magnitude slower than that of Zhou et al. [2016].

Building on the work of Zhou et al. [2016], we propose a very simple heuristic for removing self-intersections in a soup of triangles, which we extensively tested and found to produce near-perfect results: our method handles successfully all Thingi10K models, although on the one famous instance shown in Fig. 1, we had to use a coarser-than-default rounding. We also successfully tested our approach on an ad-hoc challenging instance consisting of the boolean intersection of randomly rotated unit cubes (see Fig. 2). In terms of time efficiency, our approach handles about 17k vertices per second on average in a serial implementation.

We first review the state of the art in Section 2. We then present our heuristic in Section 3, our experiments in Section 4 and give some insight into the effectiveness of Zhou et al. [2016] and our heuristics in Section 5, before concluding in Section 6.

2 State of the art

Very few solutions have been proposed in the literature for computing an intersection-free soup of triangles that approximates a 3D mesh while ensuring that its vertices have fixed-precision floating-point coordinates.

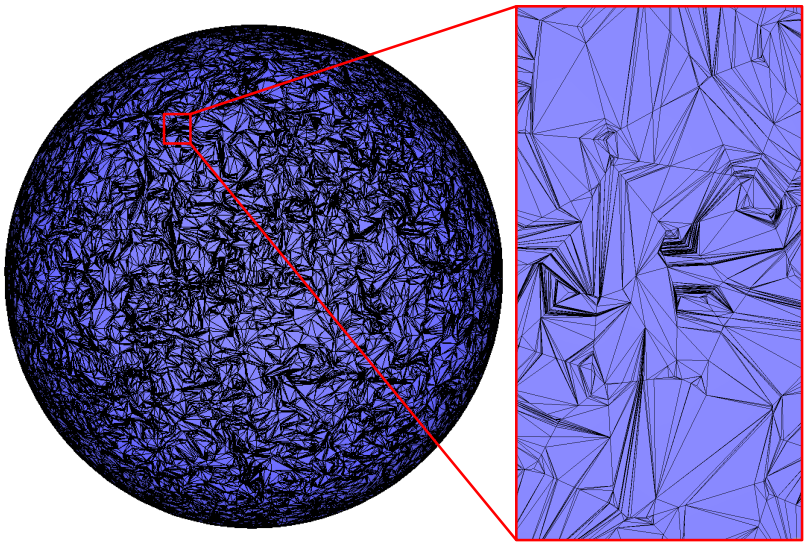


Figure 2: Boolean intersections of 1 000 randomly rotated unit cubes, all centered at the origin. The resulting model approximates the sphere inscribed in the cubes, with complex shapes on its surface.

With the notable exception of the work by Zhou et al. [2016] described below, all previous approaches start by computing the mathematically exact self-intersections in the 3D mesh and subdividing the faces accordingly, before approximating the coordinates of the vertices. Furthermore, most solutions round the coordinates onto a fixed uniform grid, usually considered to be the integer grid up to a change of scale. This problem is often referred to as *snap rounding* in the computational geometry community.

In the nineties, Milenkovic [1990] and Goodrich et al. [1997] proposed two schemes that were later proved to be flawed by Fortune [1999]. We present in Fig. 3 a counter-example to Goodrich et al.'s scheme, which highlights some difficulties of the problem: it shows two segments whose distance is realized by their endpoints (B and D) and whose projections on each of the three axis-parallel planes do not intersect, while they intersect after rounding their vertices to the centers of their grid cells; by construction, it does not help to subdivide the segments in the grid cells that realize their distance, or at the points where they intersect in projection on the axis-parallel planes.

Fortune [1999] later proposed a rounding algorithm whose grid precision depends on the number of input triangles and which is thus not usable for rounding the coordinates on a grid of fixed precision.

Zhou et al. [2016] were the first to propose an efficient practical scheme that goes beyond naive rounding. Their approach consists in (i) identifying all pairs of properly intersecting triangles (without explicitly computing their intersections), (ii) rounding the coordinates of their vertices to floats, (iii) computing all the intersections between the triangles and subdividing them accordingly, (iv) rounding all new vertices to doubles, and (v) iterating at most 20 times or until no proper intersections remain.

Zhou et al. [2016] designed their scheme for computing boolean operations and they only considered models that consist of possibly (self-)overlapping components that bound solids. However, their approach can be applied to any model when replacing the boolean operations with a computation of mesh self-intersections. Zhou et al. [2016] successfully computed intersection-free boolean self-unions for all but five of the 3 413 self-intersecting models from Thingi10K that met their cri-

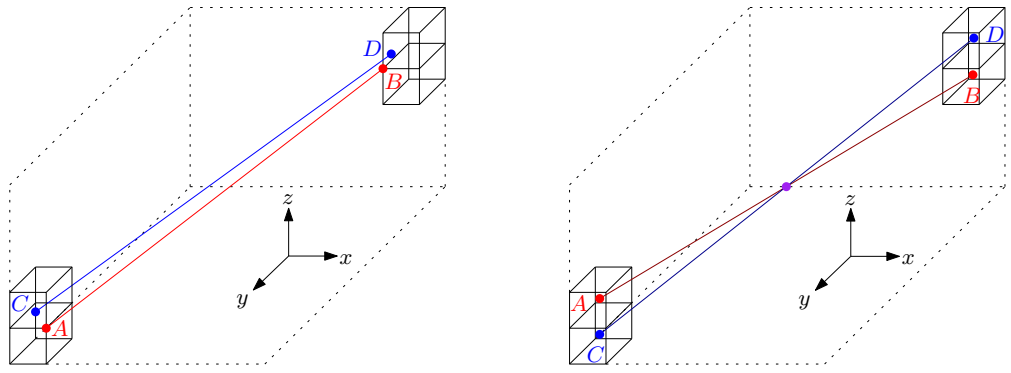


Figure 3: Counter-example to Goodrich et al. [1997]'s scheme: $A = (1, \epsilon, 1)$, $B = (6 + \epsilon, 6 + \epsilon, 4)$, $C = (\epsilon, 1, 0.8)$, $D = (6 + \epsilon, 6 + 3\epsilon, 4 + \epsilon)$ with $0 < \epsilon \ll 1$.

teria. For computing mesh self-intersections, our experiments show that on the 527 non-trivial models of Thingi10K [Zhou and Jacobson, 2016] mentioned above, their approach fails on about 9% of them (48 instances), stopping each trial after 10 hours of computation.

Milenkovic and Sacks [2019] proposed an algorithm involving iterative linear programming to increase the minimum distance between features (vertex/face and edge/edge pairs) and a post-processing using gradient descent to minimize vertex drift while maintaining feature separation. While no intersection can occur during rounding, the drawback of their approach is that vertices can be rounded arbitrarily far. They show that the vertex drift remains small in practice but their tests were mainly conducted on custom examples and not on a large data base including difficult models, making comparisons with other methods difficult.

Devillers et al. [2020] presented the first 3D snap rounding algorithm, which outputs a set of triangles with integer coordinates such that the Hausdorff distance between the input and the output is at most $\frac{3}{2}$ and the topology is preserved "up to collapse". However, the complexity of their algorithm is, in practice, too large to be usable on "real-life" models [Valque, 2019].

Building on this work, Valque [2024] proposed a practical algorithm that offers the same guarantees. Their implementation fails relatively often, that is on about 35% (187 instances) of the non-trivial models of Thingi10K. However, by relaxing some topological guarantees, their failure rate drops to about 6% (33 instances), slightly outperforming the heuristic of Zhou et al. [2016]. Nevertheless, Valque [2024]'s algorithm is much more challenging to implement and slower by two orders of magnitude than Zhou et al. [2016]'s heuristic.

3 Our approach

Our heuristic for removing self-intersections in a soup of triangles while preserving coordinates in floating-point format is as follows.

(i) Identify all pairs of properly intersecting triangles (without explicitly computing their intersections),
(ii) round their vertices to the centers of their respective cells in a uniform grid of suitable coarseness (see below),
(iii) also round all vertices that are located in these cells.
(iv) Compute all the intersections between the triangles, using exact arithmetics, and subdivide

them accordingly.

(v) Round all new vertices to doubles.

(vi) Iterate at most 5 times or until no proper intersections remain.

When the input coordinates are given as doubles, the default uniform grid we use in Steps (ii) and (iii) is the finest possible uniform grid on which the rounded coordinates of all the vertices are exactly representable as floats. In other words, for each axis, we consider a scaling by a power of two such that the largest absolute value of all the vertices coordinates lies within $[2^{23}, 2^{24})$, which is the largest scaling so that the integral part of every coordinate is representable as float; the rounding a of coordinate then amounts to applying this scaling, rounding to the closest integer and applying the reverse scaling. This rounding induces a relative error of at most $2^{-24} \approx 6.10^{-8}$, which induces a Hausdorff distance between the input and output of at most this relative error times the maximum absolute value of the coordinates times the number of iterations in our scheme.

We provide, in Section 5, insights into the effectiveness and differences between the heuristics of Zhou et al. [2016] and ours. However, we begin by presenting our experiments in Section 4.

4 Experiments

In this section, we present experiments on our heuristic and, for comparison, on naive rounding, iterative naive rounding, and Zhou et al. [2016]'s heuristic. We do not provide a detailed comparison between our heuristic and Valque [2024]'s algorithm because the latter is challenging to implement and its reported running time (processing about 100 vertices per second on average) is about two orders of magnitude slower than ours, with a lower success rate.

Most of our experiments are conducted on the Thingi10K dataset [Zhou and Jacobson, 2016] of 10 000 meshes, which were created using objects from the model-sharing website for 3D printing called Thingiverse (`https://www.thingiverse.com`). This dataset reflects the variety, complexity, and quality (or lack thereof) of 3D models. Three models in Thingi10K are unreadable[1] and were excluded from our tests. Of the remaining models, 4 524 exhibit self-intersections and were the primary focus of our experiments. We present in Figure 4 and Table 1 results on 10 pertinent models of various sizes from Thingi10k.

We present our experimental setup in Section 4.1. We analyze the success rates of the considered heuristics on the Thingi10K models in Section 4.2, their running times in Section 4.3, and their output sizes in Section 4.4.

The famously difficult model of Thingi10K shown in Fig. 1 is discussed separately, in Section 4.5, because it requires a coarser-than-default grid and because the size of the arrangement obtained by computing all self-intersections in the input model is of the charts. In Section 4.6, we consider a challenging ad hoc experiment involving the boolean intersections of many randomly rotated cubes. We finally give a summary of these experiments in Section 4.7.

4.1 Experimental setup

Our experiments were performed on a cluster network using CPUs Intel Xeon Gold 5220 with 18 cores. To simplify comparaisons, our experiments used only one core and did not use parallelization.

We consider an experiment failed or unfinished for a model if the last rounded version still presents self-intersections, or if the implementation does not terminate in less than the time limit.

In all experiments, we resolve self-intersections using CGAL library [The CGAL Project, 2024]. We use exact arithmetic for the predicates to ensure correct results in the intersection tests; this

[1] These three models are referenced as 49 911, 81 313, and 77 942.

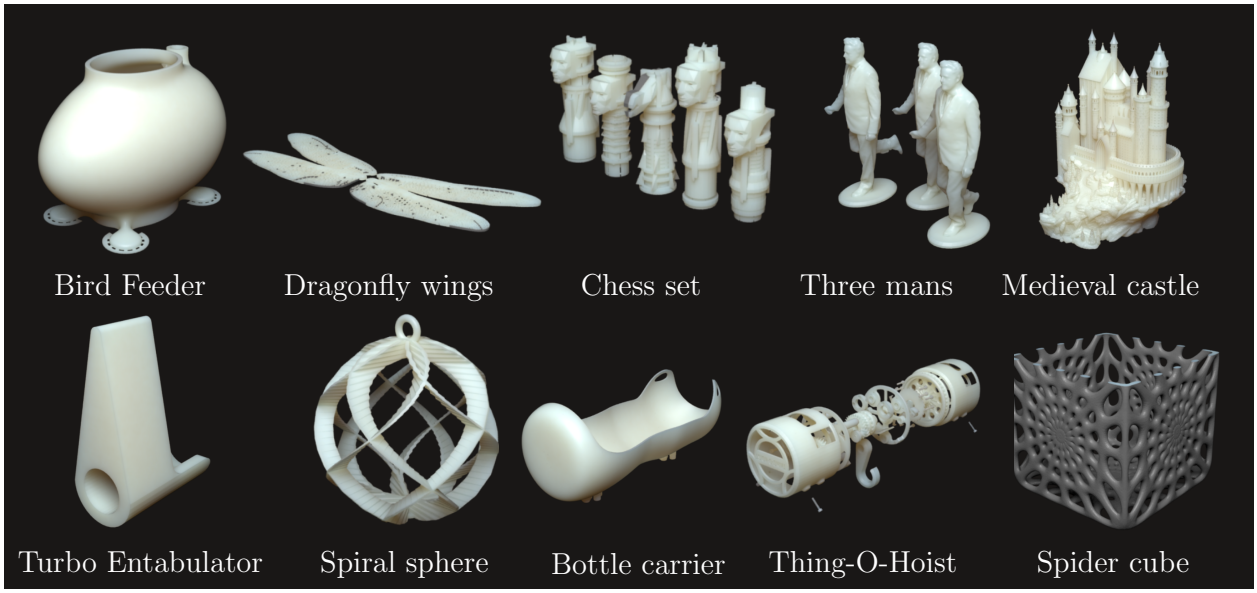


Figure 4: Ten pertinent meshes from Thingi10k [Zhou and Jacobson, 2016], we considered for the experiments in Table 1.

Input model				It. naive			Zhou et al.			Ours		
Name	**Id**	**Input V**	**Arr. V**	**It.**	**Out. V**	**Time**	**It.**	**Out. V**	**Time**	**It.**	**Out. V**	**Time**
Bird feeder	622327	125k	157k	17	138k	3min20	1	138k	9s	1	138k	9s
Dragonfly wings	498460	95k	412k	5	412k	8min26	2	412k	2min12	1	361k	1min
Chess set	113906	24k	93k	>6	-	>1h	5	92k	55s	1	91k	10s
Three mans	199665	506k	507k	>8	-	>1h	1	507k	20s	1	507k	20s
Medieval castle	1368052	1.2M	2.1M	>5	-	>1h	6	2.1M	40min	3	2.5M	21min
Turbo Entabulator	217027	528	553	>19	-	>1h	>15	-	>1h	1	524	<0.1s
Spiral Sphere	356074	14k	14k	>6	-	>1h	>19	-	>1h	1	14k	<1s
Bottle Carrier	247516	90k	197k	>4	-	>1h	>6	-	>1h	1	196k	42s
Thing-O-Hoist	71377	502k	506k	>12	-	>1h	>19	-	>1h	1	506k	22s
Spider cube	252786	355k	2.2M	>3	-	>1h	>2	-	>1h	1	2.1M	7min34

Table 1: Experiments on 10 pertinent examples of various sizes from Thingi10k : two for which the iterative naive rounding terminates, three more on which Zhou et al. [2016]'s approach terminates, and five for which they do not. Input V., Out. V and Arr. V refer, respectively, to the input and output number of vertices and to the number of vertices in the arrangement obtained by computing all self-intersections in the input model. 'It.' stands for iteration.

ensures that when the heuristics terminate within the bound on the number of iterations and the time limit, the output is certified to be intersection free. We also use exact arithmetic for the construction of intersection-induced vertices to control the maximum distance between the input and the output. This is achieved using the CGAL EPECK kernel (Exact Predicates Exact Construction Kernel) [The CGAL Project, 2024]. (Note that one can also use the exact predicates but inexact constructions kernel –EPICK– to achieve a certified output with faster runtime but less control on the input-output distance.)

For fair comparisons, we did our own implementation of all schemes. In particular, we use the same implementation for computing self-intersections, which is the most time-consuming step in all heuristics. These intersections are computed using the autorefine function of the Polygon Mesh Processing CGAL package [Coeurjolly et al., 2024].

4.2 Success rates

Naive rounding and iterative naive rounding. Recall that the naive rounding consists in computing self-intersections and rounding the newly created vertices to doubles. As already mentioned in the introduction, out of the 4 524 models in Thingi10K that contain self-intersections, the naive rounding fails on 527 models (≈12%). We focus most of the following experiments on these models, which we refer to as the *non-trivial models* of Thingi10K.

Iterating the naive rounding improves the success rate but still about half (273 of the 527 models) still exhibit self-intersections after one hour of computation (each). Interestingly, 237 of these models had more intersections in the last iteration than initially.

Zhou et al. [2016]'s heuristic. Recall that Zhou et al. [2016] applied their heuristic only to models that bound solids (PWN models) for computing boolean operations. In our experiments, we tested their heuristic on all Thingi10K models for computing mesh self-intersections.

Unsurprisingly, their scheme works on all 3 997 trivial Thingi10K models. Among the 527 non-trivial models, 48 did not terminate after 20 iterations or 10 hours, which is about 9% of the 527 models.

It should be stressed that the bound on the number of iterations is not a significant limitation. Indeed, among these 48 models, only five models terminates (within 10 hours each) if we remove this bound.[2] For the others, the number of intersections increases with each iteration in 37 models, and six models get stuck in a loop with a cyclic number of intersections after more than 100 iterations.

Our heuristic. Except the famously difficult model of Thingi10K shown in Fig. 1, which we discuss in Section 4.5, all Thingi10K models were successfully resolved. Only 26 models required two iterations, and only 3 required three iterations.[3]

4.3 Running time

Computing self-intersections is the most time-consuming step in all heuristics and it is output sensitive. We thus present running times in terms of the size of input arrangements, specifically, the number of vertices in the models after computing all self-intersections.

[2]These models are referenced as 113 422, 521 600, 86 324, 128 001, 496 388 and they terminate in 21, 22, 32, 36, and 53 iterations, respectively.

[3]These models are referenced as 1 368 052, 106 838, 78 227.

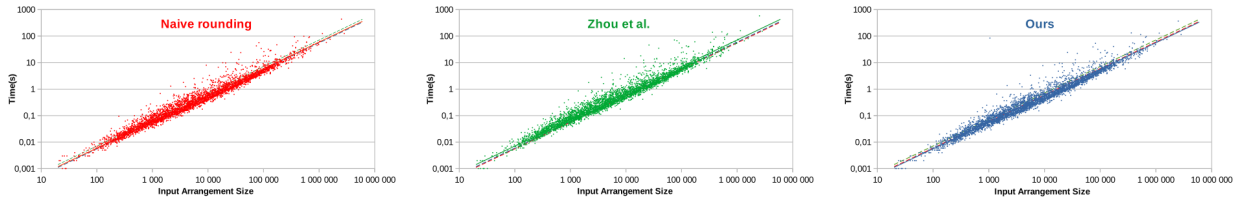


Figure 5: Running time in seconds for the 3 997 trivial models of Thingi10K in terms of the input arrangement size.

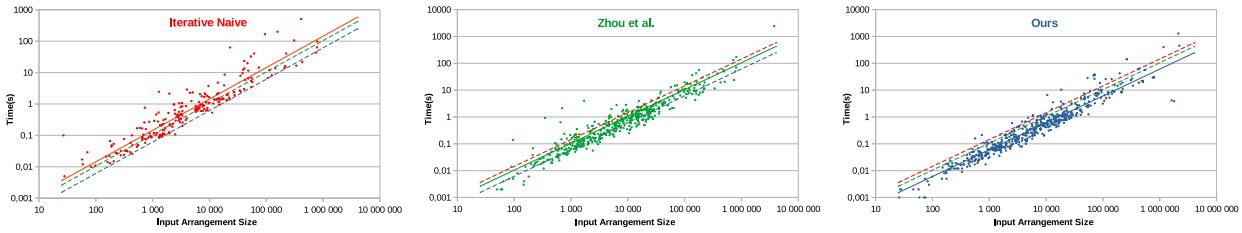


Figure 6: Running time in seconds for the 527 non-trivial models of Thingi10K (except the one of Fig. 1) in terms of the input arrangement size.

We present in Fig. 5 the running times for the 3 997 Thingi10K models for which the naive rounding works, and in Fig. 6 the running times for the other 527 non-trivial Thingi10K models (with the exception of the model shown in Fig. 1 and discussed in Section 4.5).

Data points are included only when heuristics succeed within the time bound; recall that the iterative naive rounding terminates on only 52% of the non-trivial models, and Zhou et al. [2016]'s heuristic on 91%.

For both data subsets and all schemes, the running times are well captured by linear approximations.[4] Table 2 gives the corresponding average numbers of vertices processed per second. The averages are quite similar for trivial models but for non-trivial models, the gap is larger and our scheme is on average faster than Zhou et al. [2016]'s by a factor 1.7 and faster than the iterative naive rounding by a factor 2.4.

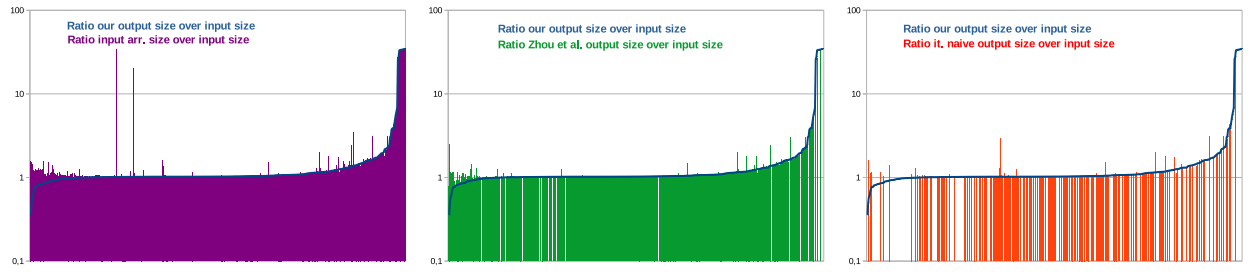
	It. Naive	**Zhou et al.**	**Ours**
Trivial models	17.5 k/s	14.7 k/s	18.9 k/s
Non-Trivial models	7.0 k/s	9.5 k/s	16.5 k/s

Table 2: Average number of input arrangement vertices handled per second.

4.4 Size of output

We present in Fig. 7(a) the ratio of the output size of our heuristic over the size of the input, and the ratio of the size of the input arrangement (i.e., after having computed self-intersections) over the size of the input, for the 527 non-trivial Thingi10K models (except the one of Fig. 1 discussed in Section 4.5).

[4]The least-square interpolations of the running time shown in Figs. 5 and 6 are the following, in milliseconds per thousand of input arrangement vertices. Trivial models: $57x$ for the naive rounding, $68x$ for Zhou et al. [2016]'s and $53x$ for ours. Non-trivial models: $144x$ for the iterative naive rounding, $105x$ for Zhou et al. [2016]'s and $60x$ for ours.



(a) (b) (c)

Figure 7: Output sizes versus input sizes. (a-c) In blue: ratio of the output size of our heuristic over the input size, for each of the 527 Thingi10K non-trivial models (except the one of Fig. 1) ordered increasingly. (a) In purple: ratio of the input arrangement size over the input size. (b) In green: ratio of Zhou et al. [2016] output size over the input size. (c) In red: ratio of the iterative naive output size over the input size. (b-c) Missing data corresponds to instances on which heuristics fails.

This shows that the size of the input arrangement is a good predictor of the size of the output, especially when the input model contains many self-intersections (i.e., when the input arrangement is larger than the input size). However, we can also observe that, occasionally, our output is much smaller than the input arrangement, the converse being very rare.

Fig. 7(b) shows the ratio of Zhou et al. [2016] output size over the input size, compared to ours and similarly for Fig. 7(c) with the iterative naive rounding scheme. These figures show that these heuristics fail more often when the input arrangement is large compared to the input size. However, when they succeed, the output size tend to be similar to ours, although occasionally larger.

4.5 The model of Fig. 1

The model shown in Fig. 1, referenced as number 996 816 in Thingi10K, is notoriously difficult due to its incredibly high number of intersections and has caused failures in many various implementations. It contains about 76k vertices and 171k triangles. Its input size is thus quite reasonable, but about 86k of its triangles are involved in about 5M pairs of intersecting triangles and 20M of triplets of intersecting triangles. Computing all self-intersections is thus already a tremendous work and the resulting exact arrangement is huge with about 30M vertices and 240M faces.

Our heuristic fails to resolve this model with our default grid size for rounding the vertices of the triangles involved in self-intersections. Recall that our default grid is obtained by scaling the uniform integer grid, where the reverse scaling (by a power of two) maps the largest absolute value of all vertex coordinates to the range $[2^{23}, 2^{24})$.

However, we successfully resolve this model in 38s (serial) by rounding the vertices on a coarser grid, namely by scaling the largest absolute value of the coordinates to the interval $[2^{13}, 2^{14})$ instead of $[2^{23}, 2^{24})$. With this coarser grid, the rounding induces a relative error of at most $2^{-14} \approx 6.10^{-5}$ in the coordinates and the L^∞ Hausdorff distance between the input and output models does not exceed 6.10^{-3} (because rounded coordinates do not exceed 100 in absolute value and the heuristic resolves this instance in only one iteration).

Furthermore, the output model contains approximately 100k vertices and 245k faces, which is satisfactory, as it is of the same order of magnitude as the input model.

It should be noted that, by increasing the precision of the rounding by a factor 2, the heuristic still resolves this model in 2min with about 280k vertices in the output. Increasing again the

precision by a factor two, the heuristic still works but in 35min and a huge output with 4.8M vertices. Increasing again the precision by a factor two, the heuristic does not terminates in the one hour time limit.

4.6 Rotated cubes

We tested the heuristics on an ad hoc challenging model that consists of the boolean intersection of many randomly rotated unit cubes, all centered at the origin and added one by one. After many iterations, the model approximates a sphere with complex shapes on its surface (see Fig. 2).

We conducted these experiments using the same initial seeds for each heuristic to ensure reproducibility, performing 16 distinct trials per heuristic with seeds ranging from 1 to 16. We computed boolean intersections using the CGAL corefine_and_compute_intersection function [Coeurjolly et al., 2024].

Each trial run was stopped at a certain iteration either if (i) the heuristic failed to remove self-intersections, or (ii) the result of the boolean intersection was not 2-manifold (a requirement of the CGAL boolean operator).

Understanding why boolean intersections of such cubes might not result in a 2-manifold is not immediately clear. A typical example, depicted in Fig. 8(c), is when the rounding scheme creates new intersections, then the resulting model of the current iteration is not 2-manifold and its boolean intersection with a cube in the following iteration is possibly not 2-manifold either.

The iterative naive rounding trials stopped after an average of 150 cubes with a standard deviation (SD) of 96, while Zhou et al. [2016]'s trials stopped after an average of 288 cubes (SD = 163). In contrast, our heuristic reached an average of 984 cubes (SD = 368), with 9 trials still in progress, each taking less than three minutes per iteration.

4.7 Summary of experiments

In terms of success rate, naive rounding fails quite often, that is on 527 (≈12%) of the self-intersecting models of Thingi10K, which are referred to as the non-trivial models of Thingi10K. The iterative naive rounding solves only about half of these non-trivial models. Zhou et al. [2016]'s heuristic is generally effective but shows limitations, as it fails on about 9% of the non-trivial models. Our heuristic successfully resolves all Thingi10K models, although its famously difficult model shown in Fig. 1 needs to be handled with a coarser-than-default grid.

In terms of efficiency, the output size is fairly well approximated by the size of the input arrangement (i.e., the input model after having computed self-intersections) and the running time of all heuristics are well approximated by linear interpolations. Our scheme handles on average at least 16k vertices of the input arrangement per second (in a serial implementation). The averages are quite similar for all heuristics on trivial models but on non-trivial models, our scheme is on average faster than Zhou et al. [2016]'s by a factor 1.7.

In the challenging ad hoc experiment of boolean intersections of randomly rotated unit cubes, all centered at the origin, previous heuristics fail with less than 300 cubes, averaged over 16 trials. In contrast, our heuristic reached an average of 984 cubes, with more than half the trials still in progress, each taking less than three minutes per iteration.

5 Insights on Zhou et al. [2016]'s scheme and ours

The effectiveness of Zhou et al. [2016]'s scheme and ours is not obvious. We present here our understanding, although supporting evidence remains elusive. (Note that Zhou et al. [2016] did

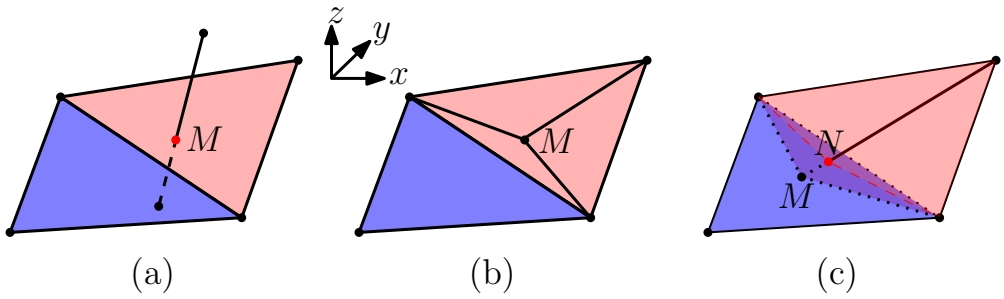


Figure 8: A typical intersection induced by rounding. (a) Edge-face intersection. (b) Subsequent face triangulation. (c) Edge-face intersection N induced by the rounding of M and the "flip" of a triangle.

not address this aspect in their paper.)

First, coarsely rounding coordinates tends to merge nearby vertices, eliminating triangles with tiny edges. This effect can be observed in the size of the output (see Fig. 7), which is occasionally much smaller than the size of the input after computing self-intersections. However, this alone does not explain the schemes effectiveness because it solves less troubles than it causes (see e.g. Fig. 8). Experiments show that rounding *all* coordinates to floats (after computing self-intersections) performs very poorly and is even worse than naive rounding: this approach fails to remove self-intersections on about 42% of the 4 524 Thingi10K models that contain self-intersections (instead of 12% for naive rounding) and iterating with a one-hour limit only reduces the failure ratio to about 24%. (Results are similar if we round coordinates on a uniform grid instead of the grid of floats.)

Consider now a (proper) intersection of an input segment with an input triangle (see Fig. 8). A simple calculation shows that each coordinate of the intersection point is a rational whose numerator and denominator are polynomials of degree 4 and 3, respectively, in the input coordinates. Further straightforward computations show that, if any two input coordinates are either equal or farther apart than δ, and if they are bounded by $\pm M$, then in projection on each of three axis-parallel planes, the distance between the intersection point and the edges of the triangle is at least $c\delta^5/M^4$, for some constant c.

This implies that, if, before computing the intersection between input triangles, we round input coordinates so that they collapse or become farther apart than δ, then, after computing the intersections, if we round vertices on a grid with a minimum spacing of $c\delta^5/M^4$, the triangles resulting from the subdivisions do not flip during rounding, in projection on the axis-parallel planes (we consider for simplicity flips in projection in the axis-parallel planes but the principle is similar in 3D).

In our scheme, we first round the vertices of the features that intersect to the finest possible uniform grid in which rounded coordinates are representable by floats. Then we round the coordinates of the intersections points on doubles, which is everywhere at least 2^{29} times finer than the uniform grid. This is not sufficient to ensure that the triangles never flip, however, the discrepancy between the two grids makes it unlikely for them to flip in practice.

While we consider a uniform grid in our scheme, Zhou et al. [2016] round the coordinates directly on floats. This implies that, in their scheme, the minimum distance between coordinates is much smaller when the coordinates are close to zero than when they are away from zero. As a consequence, the above argument, claiming that triangles rarely flip, does not hold anymore. Experimental evidence supporting this analysis is that many Thingi10K models on which Zhou

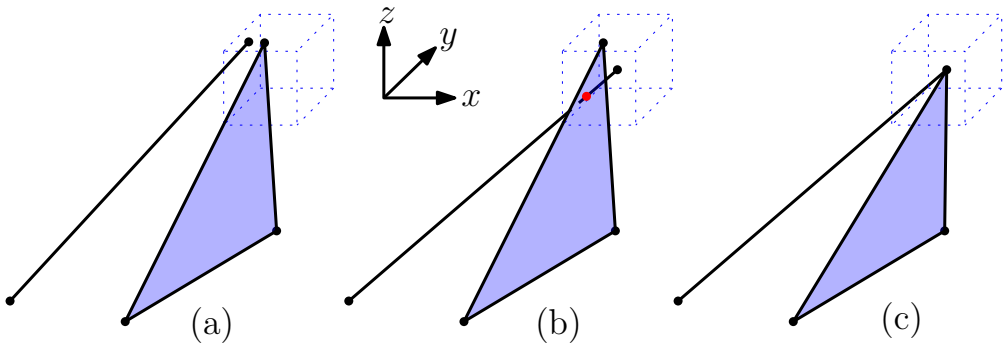


Figure 9: Another typical intersection induced by rounding. (a) Input. (ii) Intersection induced by the rounding of Step (ii). Removal of the intersection induced by the rounding of Step (iii).

et al. [2016]'s scheme fails are successfully resolved when the models are translated so that all coordinates are larger than one.

Finally, Step (iii) in our scheme, which rounds all vertices that are located in a grid cell where a vertex has already been rounded is quite natural, as it prevents additional intersections, as depicted in Fig. 9(b), with no apparent drawbacks. Without Step (iii), our scheme fails on one Thingi10K model other than the famous one of Fig. 1; this model[5] is relatively small, containing 15k vertices and 75 pairs of intersecting triangles, however the number of intersections increases significantly with each iteration. Furthermore, without Step (iii), many models (46 in total) require more iterations, which results in slower running times.

6 Conclusion

We have introduced a new scheme to address the mesh intersection problem, with a particular focus on the intersections resulting from the conversion of the vertices coordinates from their exact mathematical values to a fixed-precision floating-point format. Our evaluations on the Thingi10K dataset, where we successfully resolve all models, and on a challenging ad hoc experiment, demonstrate that our method is highly effective, even on extremely difficult instances, and significantly outperforms previous approaches.

References

Marco Attene. Indirect predicates for geometric constructions. *Computer-Aided Design*, 126:102856, 2020.

Gianmarco Cherchi, Marco Livesu, Riccardo Scateni, and Marco Attene. Fast and robust mesh arrangements using floating-point arithmetic. *ACM Transactions on Graphics (TOG)*, 39(6): 1–16, 2020.

Gianmarco Cherchi, Fabio Pellacini, Marco Attene, and Marco Livesu. Interactive and robust mesh booleans. *ACM Transactions on Graphics (TOG)*, 41(6), November 2022. ISSN 0730-0301. doi: 10.1145/3550454.3555460. URL `https://doi.org/10.1145/3550454.3555460`.

[5]This model is referenced as 105 867.

David Coeurjolly, Jaques-Olivier Lachaud, Konstantinos Katrioplas, Sébastien Loriot, Ivan Paden, Mael Rouxel-Labbé, Hossam Saeed, Jane Tournois, and Ilker O. Yaz. Polygon mesh processing. In *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition, 2024. URL `https://doc.cgal.org/6.0.1/Manual/packages.html#PkgPolygonMeshProcessing`.

Olivier Devillers, Sylvain Lazard, and William Lenhart. Rounding meshes in 3D. *Discrete and Computational Geometry*, 64(1):32–67, April 2020. doi: 10.1007/s00454-020-00202-2. URL `https://inria.hal.science/hal-02549290`.

Lorenzo Diazzi and Marco Attene. Convex polyhedral meshing for robust solid modeling. *ACM Transactions on Graphics (TOG)*, 40(6):1–16, 2021.

Steven Fortune. Vertex-rounding a three-dimensional polyhedral subdivision. *Discrete & Computational Geometry*, 22(4):593–618, 1999. doi: 10.1007/PL00009480.

Michael T. Goodrich, Leonidas J. Guibas, John Hershberger, and Paul J. Tanenbaum. Snap rounding line segments efficiently in two and three dimensions. In *Proceedings of the thirteenth annual symposium on Computational geometry*, pages 284–293. ACM, 1997. doi: 10.1145/262839.262985.

Bruno Lévy. Exact predicates, exact constructions and combinatorics for mesh csg, 2024. URL `https://arxiv.org/abs/2405.12949`.

Marco Livesu. Advancing Front Surface Mapping. *Computer Graphics Forum*, 43(2):e15026, 2024. ISSN 1467-8659. doi: 10.1111/cgf.15026.

Victor Milenkovic. Rounding face lattices in d dimensions. In *Proceedings of the 2nd Canadian Conference on Computational geometry*, pages 40–45, 1990.

Victor Milenkovic and Elisha Sacks. Geometric rounding and feature separation in meshes. *Computer-Aided Design*, 108:12–18, 2019. ISSN 0010-4485. doi: https://doi.org/10.1016/j.cad.2018.10.003. URL `https://www.sciencedirect.com/science/article/pii/S0010448518302896`.

The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 6.0.1 edition, 2024. URL `https://doc.cgal.org/6.0.1/Manual/packages.html`.

Philip Trettner, Julius Nehring-Wirxel, and Leif Kobbelt. Ember: exact mesh booleans via efficient & robust local arrangements. *ACM Trans. Graph.*, 41(4), July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530181. URL `https://doi.org/10.1145/3528223.3530181`.

Leo Valque. 3D Snap Rounding. Master's thesis, Université de Lyon, June 2019. URL `https://hal.inria.fr/hal-02393625`.

Léo Valque. *3D Snap Rounding*. PhD thesis, Université de Lorraine, Décember 2024.

Qingnan Zhou and Alec Jacobson. Thingi10k: A dataset of 10,000 3d-printing models, 2016. URL `https://arxiv.org/abs/1605.04797`.

Qingnan Zhou, Eitan Grinspun, Denis Zorin, and Alec Jacobson. Mesh arrangements for solid geometry. *ACM Transactions on Graphics (TOG)*, 35(4):1–15, 2016.