Graphical Models 76 (2014) 658–668

Contents lists available at ScienceDirect

Graphical Models

journal homepage: www.elsevier.com/locate/gmod

Direct repair of self-intersecting meshes ☆

Marco Attene *

Consiglio Nazionale delle Ricerche, Istituto di Matematica Applicata e Tecnologie Informatiche, Genova, Italy

ARTICLE INFO

Article history:
Received 11 June 2014
Received in revised form 19 August 2014
Accepted 15 September 2014
Available online 30 September 2014

Keywords:
Outer hull
Mesh repairing
Self-intersection

ABSTRACT

A fast and exact algorithm to eliminate intersections from arbitrary triangle meshes is presented, without any strict requirement on the input. Differently from most recent approaches, our method does not rely on any intermediate representation of the solid. Conversely, we directly cut and stitch mesh parts along the intersections, which allows to easily inherit possible surface attributes such as colors and textures. We rely on standard floating point arithmetics whenever possible, while switching to exact arithmetics only when the available fixed precision is insufficient to guarantee the topological correctness of the result. Our experiments show that the number of these switches is extremely low in practice, and this makes our algorithm outperform all the state-of-the-art methods that provide a guarantee of success. We show how our method can be exploited to quickly extract the so-called *outer hull* even if the resulting model is non manifold by design and the single parts have boundaries, as long as the outer hull itself is unambiguously definable.

1. Introduction

Modern 3D modeling paradigms enable extremely flexible tools to design original shapes by composition and deformation of existing models [17,4]. Unfortunately, most of the times the designer has not a deep knowledge of the underlying mathematics and simply produces an original shape by cutting, bending and moving parts of 3D objects in space (e.g. Fig. 1(left)). The designer's work terminates when the object looks as expected, but in most cases the so-constructed model has a number of self-intersections that make it unusable in important applications such as, e.g., 3D printing [13,37]. Furthermore, besides self-intersections, such compound models often contain other flaws such as open boundaries, degenerate elements and singularities that complicate the necessary repairing task. As observed in [3], each repairing algorithm has its own requirements on the input, and these requirements are not always met in practice. In particular, current solutions to convert self-intersecting meshes to valid polyhedra are either too slow and memory-intensive, or too restrictive in terms of both supported input and potential output (see Section 5.1.6 in [3]). As an example of supported input restriction, one may consider that many algorithms assume that the input has a manifold connectivity and has no surface holes. In most cases, these same algorithms can only produce manifold results, which means that a nonmanifold union of two manifold solids cannot be constructed. Tools to fill holes and convert a generic input to a combinatorial manifold exist [19], but in some cases performing such a preprocessing may become quite complicated, while in some other cases (e.g. when non-manifold configurations are designed on purpose [1]) such a conversion cannot be realized at all without corrupting the original design intent.

Herewith we present a fast and exact algorithm to eliminate intersections from arbitrary triangle meshes, without any strict requirement on the input. Also, we show how

☆ This paper has been recommended for acceptance by Tao Ju and Peter Lindstrom.

* Fax: +39 0106475660.
E-mail address: marco.attene@ge.imati.cnr.it

http://dx.doi.org/10.1016/j.gmod.2014.09.002
1524-0703/

our method can be exploited to extract the so-called *outer hull* even if the resulting model is non manifold by design and the single parts have boundaries. We argue that when a designer says that "the object looks as expected" he/she does not care of non-visible parts. Thus, our outer hull extraction can be considered as an effective repairing that does not introduce any visible shape distortion. Though some state-of-the-art methods could be exploited to resolve this problem, our algorithm represents the currently fastest solution thanks to a novel mixed use of finite precision and exact arithmetics.

2. State of the art

A mesh repairing algorithm can be based on either a local or a global approach [24,3]: in the former case, local operations are performed around each defect to correct it, whereas the remaining intact parts of the mesh are kept unaltered. Conversely, in the latter case the mesh is completely rebuilt starting from an intermediate representation. Thus, highly-detailed, feature-rich meshes with mainly isolated flaws should be fixed using local approaches to preserve as many details as possible, while highly corrupted or inconsistent meshes with multiple types of defects (e.g. polygon soups) would better be fixed through a global approach, especially if one needs a guarantee that the repair process succeeds. Global approaches typically are highly robust whereas local approaches are less invasive. A few approaches exist that try to couple guarantee of success and minimal invasiveness [28,29], but they cannot treat self-intersecting meshes. On the other hand, local algorithms that fix self-intersecting meshes exist, but they cannot treat a generically flawed input. The state of the art in self-intersection repairing can be subdivided in fragile, approximate and robust methods.

2.1. Fragile algorithms

Earliest approaches to calculate and remove mesh intersections were designed to compute boolean operations [25], and the fact that robustness issues may easily arise was already known [21]. Instead of handling all the possible particular cases, in [33] BSP trees are used as an intermediate representation, and an improved version of this approach has been proposed by [27]. However, instead of solving the robustness problem, algorithms of this kind can just detect too difficult situations and report a generic failure. Surprisingly, such a "fragile" approach is quite diffused in current commercial systems (e.g. Autodesk Maya [6] and 3ds Max [5]), and when the algorithm fails the user is simply allowed to undo the operation and tweak the model before retrying.

2.2. Approximate methods

Instead of trying to compute an exact solution using a fragile approach, another class of methods achieves robustness at the cost of providing an approximated solution. As observed in [9], tessellated CAD models produced by designers are typically made of many surface patches that should perfectly adhere along their boundaries. Due to diverse sources of approximation, however, these adjacency lines are often replaced by gaps or self-intersections. The solution proposed in [9,35] is to rebuild the mesh in a neighborhood of these defects through a local reconstruction. Clearly, this approach is less invasive than a global remeshing such as in [23], but the result provided is an approximation in any case. Furthermore, in [9] each single patch in the input is required to be free of self-intersections. A more general method is presented in [10], where an octree is used to completely resample the input, so that the result is guaranteed to be the boundary of a solid object which stays within a user-prescribed distance from the original mesh. When the input is a high resolution raw digitized mesh, self-intersections can be removed using [2] even if the mesh has degenerate facets, surface holes and singularities. Though being extremely robust and editing the mesh only locally, this method and all the others discussed in this subsection cannot produce exact solutions (i.e. exact outer hulls).

2.3. Robust and exact methods

The easiest way to implement algorithms which are both robust and exact is to rely on exact arithmetics or similar techniques. Exact geometric *predicates* may be used by

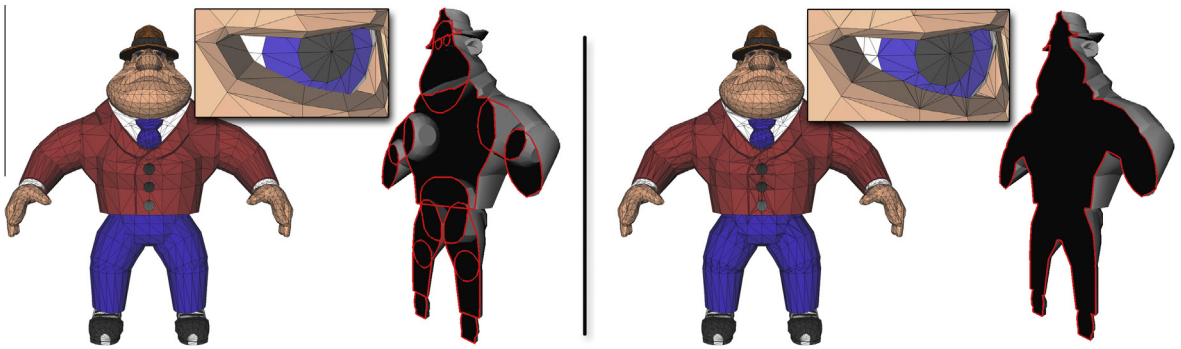

Fig. 1. A typical example of compound model made by juxtaposition of simpler parts (left). Although the external aspect may be satisfactory for a designer, such a model is not suitable for many applications. Our algorithm can effectively convert such a collection of intersecting parts, along with possible attributes (e.g. colors and textures), into a valid model that can be exploited in a much larger set of scenarios (right, triangles are consistently connected to form a single surface). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)