Computer-Aided Design 45 (2013) 288–300

Contents lists available at SciVerse ScienceDirect

Computer-Aided Design

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

Direct rendering of Boolean combinations of self-trimmed surfaces

Jarek Rossignac [a], Ioannis Fudos [b,*], Andreas Vasilakis [b]

[a] School of Interactive Computing, College of Computing, Georgia Tech, Atlanta, GA 30332, USA

[b] Department of Computer Science, University of Ioannina, GR45110 Ioannina, Greece

ARTICLE INFO

Keywords:
Rendering
CAD
Trimming
GPU
Capping
Clipping
Animation
Solid modeling
CSG

ABSTRACT

We explore different semantics for the solid defined by a self-crossing surface (immersed sub-manifold). Specifically, we introduce rules for the interior/exterior classification of the connected components of the complement of a self-crossing surface produced through a continuous deformation process of an initial embedded manifold. We propose efficient GPU algorithms for rendering the boundary of the regularized union of the interior components, which is a subset of the initial surface and is called the trimmed boundary or simply the *trim*. This classification and rendering process is accomplished in real time through a rasterization process without computing any self-intersection curve, and hence is suited to support animations of self-crossing surfaces. The solid bounded by the trim can be combined with other solids and with half-spaces using Boolean operations and hence may be capped (trimmed by a half-space) or used as a primitive in direct CSG rendering. Being able to render the trim in real time makes it possible to adapt the tessellation of the trim in real time by using view-dependent levels-of-detail or adaptive subdivision.

1. Introduction

Rendering boolean combinations of self-crossing surfaces in real time is central to a number of applications such as animation of deforming objects, preview of CAD operations and collision detection. FFD (free-form boundary deformations) is a popular paradigm for designing 3D shapes. FFD affords an intuitive direct manipulation and seems most appropriate for editing medical and artistic models or animations. The designer may for example use 3D input devices to grab, pull, and twist the 3D model in natural and predictable ways to create self-intersecting surfaces [1]. Unfortunately, FFD lacks a useful semantics of what happens when the designer wishes to create a self-intersecting surface model. One may argue that in a static model, the user could be asked to select which manifold portions of the surface should be removed by clicking on them. We offer a semantics that makes this selection process unnecessary. More importantly, if we want to apply these topological changes to an animated model, we cannot expect the user to perform these selections at each frame. We need a semantics for performing these selections automatically in a manner that is coherent over time and that is compliant with the results that would be obtained through CSG operations. This paper introduces a framework for treating self-trimmed surfaces as first class citizens, allowing us to use them as CSG primitives

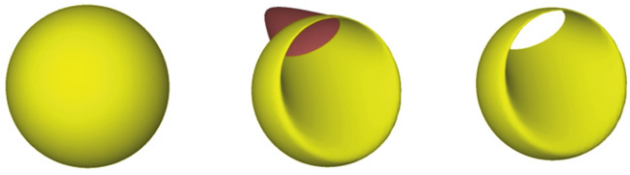

Fig. 1. We show the original manifold boundary S_0 (left), a frame S_t produced by continuously deforming S_0 (we use pink to illustrate the part of the surface that should be trimmed) (center) and the trimmed result $T(S_t)$ (right).

or to show their cross-sections (intersections with a plane) using capping. In this work, we formally define the problem, explore rules that capture application semantics and provide efficient GPU-rendering algorithms.

Formally, a surface S is **manifold** when it is a compact and orientable two-manifold without boundary. We say that S is **self-crossing** when its immersion contains non-manifold self-intersection edges where S "passes through itself", as shown in Fig. 1. Hence, two or more different points of S coincide at each point of a self-crossing curve of the immersion. We say that S is a **boundary** when S is the boundary of some solid (closed-regularized point set) that we denote $I(S)$ and call the **interior solid** of S.

Consider an initial manifold boundary S_0 that is not self-crossing and a continuous process D_t that deforms this surface while keeping it an immersed sub-manifold. Let S_t denote the instance $D_t(S_0)$ of the deformed surface at time t. Assume that S_t is self-crossing, then we say that S_t is a **Self-Crossing Surface**,

* Corresponding author.
E-mail address: fudos@cs.uoi.gr (I. Fudos).

0010-4485/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2012.10.012

abbreviated *SCS*, i.e., a compact, immersed, and orientable surface with transverse self-intersections. An example of the above setting is illustrated in Fig. 1.

We explore here different semantics for defining the solid that it represents, which we call its interior $I(S_t)$, and hence also its **trim**, $T(S_t)$, which is the subset of S_t that is the boundary of $I(S_t)$. We say that $T(S_t)$ is a **Self-Trimmed Surface**, abbreviated *STS*. The interior along with the STS define a manifold or a non-manifold object.

In this context we identify two generic problems and devise rules to capture them:
Problem I: Given an SCS (Fig. 1(b)) determine the trim (Fig. 1(c)).
Problem II: Given an initial manifold boundary S_0 (Fig. 1(a)) and a continuous process D_t that deforms this surface to an SCS S_t (Fig. 1(b) determine the trim $T(S_t)$ (Fig. 1(c)).

To address the first problem, we explore static rules that depend only on the SCS and evaluate at least in simple cases how well the results they produce match what we consider to be plausible intentions of the designer.

The second problem corresponds to dynamic rules that depend on the deformation history and the SCS. We are particularly interested in formulations of $T(S_t)$ that correspond to a designer's intuitive expectation of the sequence of results that should be produced by a reasonable deformation D_t that creates several self-crossings. In particular, we propose semantics that mimic locally the natural behavior of incremental Boolean operations, where self-crossings are created in S_t one at a time and each performs a local union or intersection of shapes defined partially by two portions of the previous frame S_{t-1}.

We also introduce practical and efficient GPU-based **trimming** algorithms that render $T(S_t)$ directly by scan-converting S_t and S_{t-1} (where t is the frame number) without the need for computing self-intersection curves. We do this by testing surfels, to establish whether they lie on the boundary of $I(S_t)$. Surfels are represented by fragments of the surface that arise from the intersection of the surface with a ray originated at the center of the corresponding pixel of the viewing plane.

We claim three advantages of such a direct trimming and rendering approach. The first advantage is the elimination of the cost of computing self-intersection curves and of identifying the faces (connected components that are cut out by these curves). Such a cost would otherwise make it impossible to render the trim during deformation animations or perform interactive editing. The second advantage is the flexibility of being able to define S_t as the result of a (possibly adaptive) subdivision process to be carried out on the GPU. Finally, we can render on the GPU the result of combining the interiors of two or more self-crossing surfaces through CSG operations.

2. Background and problem definition

An SCS S partitions the 3D space W into open full dimensional **components** C_i (i.e., the maximally connected components of $W - S$), one of which is infinite (denoted here by C_0). Each component is classified as either ***interior*** (also denoted as ***in)***, i.e., part of the interior of the solid, or ***exterior*** (also denoted as ***out)***. The solid $I(S)$ represented by S is the closure of the union of all interior components. Our objective is to define a rule that selects the components of $I(S)$. To obtain a bounded solid, C_0 should not be included, and hence is **classified** as *out*. Other components may be classified as *in* or *out*, depending on the chosen rule.

The **trim**, $T(S)$, is the boundary of $I(S)$. Hence, trimming amounts to discarding portions of S that separate either two interior or two exterior components.

Various **rules** (semantics) may be used to associate a solid $I(S)$ with an SCS S. One may conceive interesting rules that compute new bounding surfaces for solid $I(S)$ (by for instance using the convex hull of S or a visibility graph). Here, we focus on rules that have the **boundary diminishing** property, which states that the boundary $T(S)$ of $I(S)$ must be a subset of S. Note that this property is satisfied by Boolean and regularized Boolean operations [2,3].

In 2D, the **index** (also called **winding number**) $w(p, C)$ of an oriented, closed-loop, self-crossing curve C around a given point $\boldsymbol{p}$ that is not on C is an integer representing the total number of times the curve travels counter-clockwise around the point. The winding number depends on the orientation of the curve, and is by convention negative if the curve travels around the point clockwise. All points in a given component (maximally connected component of the complement of C) have the same winding number. The infinite component has winding number 0. One may easily keep track of the winding number by propagating it from one component to an adjacent one. Crossing the curve once increments or decrements the winding number, depending on the orientation of the curve relative to the direction of the crossing.

In three dimensions, the **index** $w(p, S)$ of a point p with respect to an SCS S may be defined as follows. We assume that p is not on S. Consider any **path** P from infinity to p. Let k_i be the number of times that P enters S (i.e., crosses the boundary in a direction opposite to the outward normal) and k_o be the number of times that P **exits** S (i.e., crosses the boundary in a direction confluent to the normal). Then, $w(p, S) = k_i - k_o$. Fig. 2 shows an SCS cross section (green self-crossing polyline) with the triplet $(k_i, k_o, w(p, S))$ indicated for each area, where k_i and k_o were computed from the left. Note that points of the same component may have different k_i and k_o (for example when counting from the bottom) but they have the same index.

For conciseness, we denote $w(p, S)$ by $w(p)$ or by simply w. Heisserman [4] provides an equivalent definition of the index (winding number) as the number of times the surface encloses a point.

Note that in situations where P simultaneously crosses several neighborhoods of S, the crossing of each neighborhood must be accounted for separately.

The index is the signed generalization of the *overlap count* which is defined as the unsigned count of the number of surfels that correspond to a pixel and is used in rendering to determine the transparency effect. Note that the index is in general not equal to the overlap count, which is defined at a point p as the number of times a specific ray from p to infinity hits the surface. When the ray is aimed at the viewpoint, the overlap count may be computed on the GPU for each pixel and used to control transparency effects. Note that the parity of the overlap count is independent of the direction of the ray, identical to the parity of the index, and constant throughout a component.

We want to use the fragments (depth, color, and normal values of S associated with points of T that project on a pixel center) that are generated by this rasterization to render scenes where $I(S)$ is **capped** [5,6] and combined with other shapes through CSG operations [7,8]. **Capping** returns the intersection of $I(S)$ with a 3D region R that is the intersection or the union of (usually) linear half-spaces and displays the **caps**, i.e., the portions of the boundary of R in $I(S)$.

3. Prior art

Interior Visualization. Different techniques have been proposed for visualizing the interior and hidden portions of solids or assemblies. Hidden edges and silhouettes may be overlaid with a shaded rendering of the visible surfaces [9]. Surface transparency and depth ordering [10] may be used to modulate the color based on the translucent surfaces seen through a pixel. Volume rendering may be used to modulate color based on the thickness of solid layers traversed by a ray from the viewpoint (see e.g. [11]).