Computer-Aided Design 58 (2015) 123–131

Contents lists available at ScienceDirect

Computer-Aided Design

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

Correct resolution rendering of trimmed spline surfaces☆

Ruijin Wu *, Jörg Peters *

University of Florida, United States

HIGHLIGHTS

- Tight estimates relate domain resolution to screen resolution of trimmed surfaces.
- Based on the estimates, sub-pixel accuracy of a display algorithm is proven.
- The algorithm has been implemented within the standard graphics pipeline.
- The implementation enables interactive editing of trimmed surfaces.
- The implementation has a small memory footprint.

ARTICLE INFO

Keywords:
Trimming
Spline
Accurate
Real-time
Scan density

ABSTRACT

Current strategies for real-time rendering of trimmed spline surfaces re-approximate the data, pre-process extensively or introduce visual artifacts. This paper presents a new approach to rendering trimmed spline surfaces that guarantees visual accuracy efficiently, even under interactive adjustment of trim curves and spline surfaces. The technique achieves robustness and speed by discretizing at a near-minimal correct resolution based on a tight, low-cost estimate of adaptive domain griding. The algorithm is highly parallel, with each trim curve writing itself into a slim lookup table. Each surface fragment then makes its trim decision robustly by comparing its parameters against the sorted table entries. Adding the table-and-test to the rendering pass of a modern graphics pipeline achieves anti-aliased sub-pixel accuracy at high render-speed, while using little additional memory and fragment shader effort, even during interactive trim manipulation.

1. Introduction

A standard approach to designing geometry in computer aided design is to "overfit", i.e. create spline surfaces that are larger than needed and subsequently trim the surfaces back to match functional constraints or join to other surfaces (see Fig. 1). This approach persists both for historical reasons and for design simplicity: for historical reasons in that overfitting and trimming predates alternative approaches such as subdivision surfaces [1,2] and finite geometrically-smooth patch complexes (see e.g. [3,4]); for practical reasons, in that it is often more convenient to control the shape of a signature piece in isolation than when constraints have to be taken into consideration. For example, a car's dashboard can be prepared in one piece without consideration of cut-outs for instrumentation and the steering column.

The prevailing practice in computer aided design environments is to generate and display a fixed-resolution triangulation on the CPU and transfer it to the GPU. This process interrupts the design process and can yield unsatisfactory results as closeups reveal a jagged or otherwise incorrect approximation. Conversely, an overly fine triangulation wastes resources: there is no need to highly resolve a complex trim curve when the corresponding surface measures only a few pixels on the screen.

The computer graphics community has developed a number of clever techniques, reviewed in Section 2, to deliver real-time display of trimmed spline surfaces. The present paper advances the state-of-the-art by carefully *predicting how fine an evaluation of the trim curves results in correct trim decisions at screen resolution*. This tight prediction makes it possible to construct, as a prelude to each modified view or model rendering pass, a slim and adaptive trim-query acceleration table that supports a light-weight per-fragment trim test. This simple add-on to any rendering pass is efficient enough to allow interactive trim-curve editing.

Overview. Section 2 reviews existing techniques for fast rendering of trimmed spline surfaces. Section 3 reviews basic concepts and

☆ This paper has been recommended for acceptance by Dr. Vadim Shapiro.

* Corresponding authors.
E-mail addresses: ruijin@cise.ufl.edu (R. Wu), jorg@cise.ufl.edu (J. Peters).

http://dx.doi.org/10.1016/j.cad.2014.08.012
0010-4485/

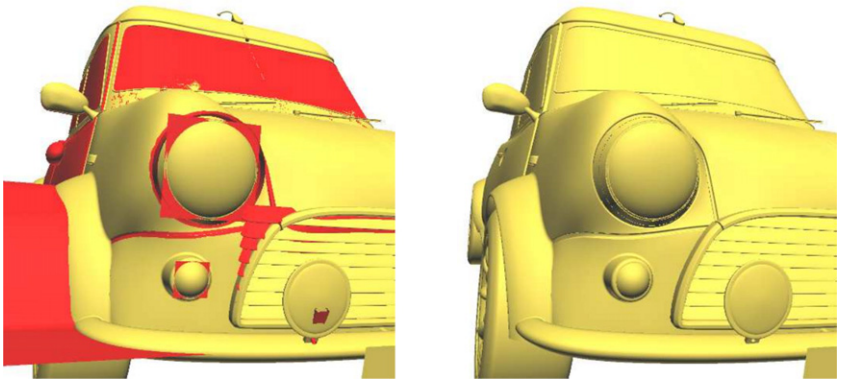

Fig. 1. Geometric design with trimmed surfaces. The red spline pieces need to be trimmed away. See also Figs. 2 and 16. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

establishes notation. Section 4 explains how correct resolution can be determined. Section 5 explains how to build and use the trim-query acceleration table. Section 6 measures the performance of a full implementation.

2. Real-time rendering of trimmed surfaces and related techniques

The *trim decision* is to determine to which side, of a set of trim curves, lies the uv-pre-image of a pixel. The underlying challenge is the same as when determining the fill region of a planar decal [5]—except that in planar filling, the accuracy of rendering is measured in the uv-plane, while for trimmed surfaces, the accuracy is measured in screen space, i.e. after applying the non-linear surface map followed by projection onto the screen.

A straightforward approach, used in 2D vector graphics [6, Section 8.7], is to *ray-test*: each pixel's uv-pre-image in the domain sends a ray to the domain boundary to determine the number of intersections (and possibly the intersection curve orientation). The intersections decide whether the fragment is to be discarded. For example, Pabst et al. [7] test scan-line curve intersection directly in the Fragment Shader. Direct testing without an acceleration structure is impractical since the number and complexity of intersections can be unpredictably high so that robustness and accuracy are difficult to assure within a fixed time. It pays to pre-process the trim curves and map them into a hierarchical search structure in order to localize testing to a single trim curve segment. For example, Schollmeyer et al. [8] break the segments into monotone pieces and test scan-line curve intersection in the Fragment Shader by robust binary search. This pre-processing is view-point independent but becomes expensive for interactive trim-curve manipulation.

An alternative is to generate a *trim texture*: the uv-pre-image of each fragment indexes into a texture that returns whether the point is to be trimmed or not. Such a trim texture can be generated in a separate rendering pass, using the stencil buffer [9]. The trim-test is highly efficient, requiring only a single texture look-up to classify a domain point. However trim textures need to be recomputed for every viewpoint change and the separate pass can noticeably lower overall performance as each segment of every trim curve generates and atomically inserts a triangle into the stencil buffer. Moreover, the trim-texture represents a uniform, limited resolution sampling of the uv-domain. Projective foreshortening must be accounted for separately: when rendering a curved surface in 3-space, the non-uniform distortion of the domain caused by the non-linear map of the surface followed by perspective projection can result in low render quality even where the texture resolution is high.

Another pre-processing choice is to convert the piecewise rational trim curves into an implicit representation, via resultants (see e.g. [10–12]). Evaluating the resultant will generate a signed number and the sign can be used to determine whether a pixel is to be trimmed. In principle, this yields unlimited accuracy. However, there are several caveats to this approach. First, the use of resultants increases the degree and the number of variables. The coefficients of the implicit representation are typically complicated expressions in terms of the coefficients of the trim curve segments. Therefore the evaluation in the Fragment Shader can be expensive even if the derivation of the implicit expression is done off-line prior to rendering. There are more efficient approaches than full implicitization, e.g. [13]. While useful for ray-tracing, these expressions do not presently yield a signed test as required for trimming. Second, implicitization converts the entire rational curve, not just the required rational *piece*. Use of resultants therefore requires a careful restriction of the test region, for example by isolating bounding triangles in the domain that contain a single indicator function whose zero level set represents the trim. Determining such restrictions is in general tricky since the implicit can have extraneous branches. For conics, the conversion expressions are sufficiently simple and for fixed shapes, such as fonts, determining bounding triangles can be done offline once and for all [14]. For less static scenarios, also pre-processing of conics and triangulation of the domain is not easily parallelized. Stencil buffers can avoid careful triangulation [5] but the fixed resolution inherits the challenges of texture-based trimming.

Computing the trim curves from CSG operations addresses a related but somewhat different problem than rendering. Here the trim curves (exact intersection pre-images) are not given. For simple CSG primitives such as quadrics the render decision can be based on an implicit in/out test. For more complex B-reps, faceted models are compared and proper resolution of the B-rep into facets remains a challenge. Practical implementations use stencil operations, depth and occlusion testing [15,16].

3. Definitions and concepts

Coordinates and projection. In the OpenGL graphics pipeline [17, Section 13.6], the non-orthogonal projection P

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \to \begin{pmatrix} x_c \\ y_c \\ z_c \\ w_c \end{pmatrix} := \begin{pmatrix} P_{11} & 0 & 0 & 0 \\ 0 & P_{22} & 0 & 0 \\ 0 & 0 & P_{33} & P_{34} \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

maps camera coordinates $(x, y, z, 1)^T$ with the camera is at the origin pointing in the negative z-direction, to clip coordinates $(x_c, y_c, z_c, w_c)^T$. The entries $P_{33} := (\overline{z} - \underline{z})/(\overline{z} + \underline{z})$ and $P_{34} := 2\overline{z}\underline{z}/(\underline{z} - \overline{z})$ define two planes at depth $\underline{z}$ (near) and $\overline{z}$ (far) such that any geometry with depth outside the range $[\underline{z}, \overline{z}]$ is clipped.