Normal Bounds for Subdivision-Surface Interference Detection

Eitan Grinspun Peter Schröder

Caltech

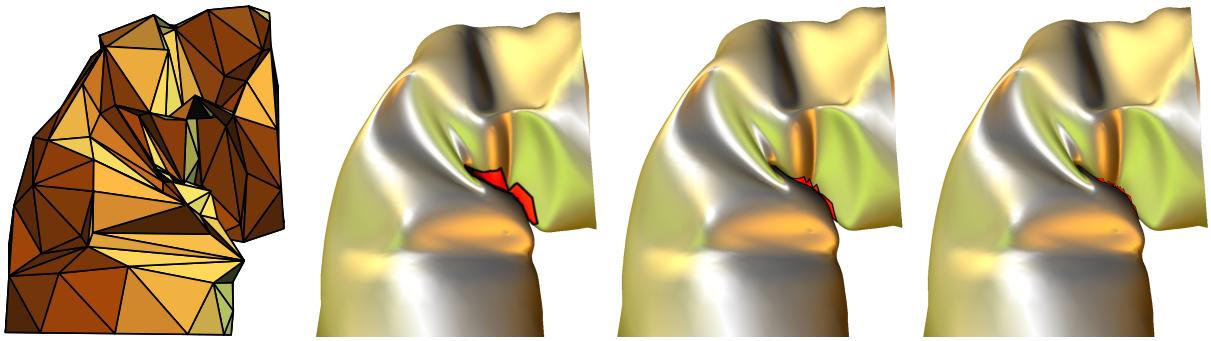

Figure 1: Collision detection in a thin-shell simulation. *Given a control mesh of a bent tube, our algorithm identifies interfering regions on the limit surface up to a user-specified precision. Above, we have resolved interference with progressively increasing precision.*

Abstract

Subdivision surfaces are an attractive representation when modeling arbitrary topology free-form surfaces and show great promise for applications in engineering design [5, 6] and computer animation [10]. Interference detection is a critical tool in many of these applications. In this paper we derive normal bounds for subdivision surfaces and use these to develop an efficient algorithm for (self-) interference detection.

CR Categories: I.3.5 [**COMPUTER GRAPHICS**]: Computational Geometry and Object Modeling - Constructive solid geometry (CSG), Geometric algorithms, Hierarchy, Splines; I.6.m [**SIMULATION AND MODELING**]: Miscellaneous - Collision detection; G.1.2 [**NUMERICAL ANALYSIS**]: Approximation - Spline and piecewise polynomial approximation

Additional Keywords: Subdivision Surfaces, Multiresolution Surfaces, Self-interference, Gauss map, Loop's Scheme

1 Introduction

Interference detection algorithms are vital for simulation and animation. As examples, consider cloth simulation [2, 8, 26] and deformable object animation [31, 32]. These applications must detect and correct self-interference and interference between surfaces. In constrained settings, when (a) the motion of objects can be expressed as a closed function of time, (b) the configuration of objects is simple or highly symmetrical, or (c) objects undergo only polynomial deformation, interference detection can be handled in simple ways through specific geometric optimizations. However, in a general setting, with complex deformations—such as wrinkling of cloth or buckling of aluminium—interference detection is difficult and time-consuming [8, 33].

Our particular interest in interference detection was born of a larger project—we simulate structures governed by the thin-shell equations, and produce physically realistic animations of crushing, crumpling, wrinkling, and other non-linear phenomena (see Figure 1). Previous publications on the Subdivision Element Method, by Cirak and other members of our team, describe the benefits of using subdivision surfaces in this setting [5, 6].

Subdivision surfaces are now widely deployed in many computer graphics and geometric modeling tasks (for an overview see [34]). They are desirable for many modeling, animation, and simulation applications, in part because they efficiently and robustly generate smooth surfaces from an arbitrary topology control mesh. However, (self-)interference detection for subdivision surfaces has never been explored and we present the first algorithm of its kind for this task. Our algorithm targets a general setting with arbitrary deformations. In particular, we make no assumptions on the

- motion of the surface,
- symmetry or simplicity of the surface configuration, or
- type of deformation applied to the surface.

To appreciate various algorithmic choices we must distinguish between *interference/collision detection* and *surface intersection*. Collision detection asks *whether* a surface coincides/intersects with itself or other surfaces. In contrast, surface intersection involves *finding* the actual intersection curves. In this paper we describe a hybrid approach tailored to simulation applications. The algorithm produces a list of interfering regions up to some user-specified resolution (see the red marked patches in Figures 1, 11, and 12), but it does not trace the exact intersection curve. Our thin-shell simulator uses this algorithm to place contact forces between abutting or interpenetrating surfaces. While the algorithm caters to simulations, its mathematical foundation is widely applicable. Our approach and math derivations apply to all popular subdivision schemes. We shall make our discussion more concrete by focusing on Loop's scheme [23].

Contributions We introduce:

- a new framework for collision and self-collision detection of subdivision surfaces undergoing arbitrary deformations, with optimizations specifically for self-collision detection;
- techniques to bound the direction of surface normals of subdivision surfaces. In particular, we address irregular patches, which *cannot* be treated with approaches reported for splines.

Our bounding technique is useful for a number of algorithms in constructive solid geometry [3], machining and milling [16, 18], simulation [5], animation [10], and trimming [22]. It is also applicable to surface area approximations, silhouette curve computations, and rendering [18]. We will however not pursue these different applications and instead focus on the basic bound construction and its use in (self-)interference detection.

1.1 Review of Related Work

Interference detection has been studied extensively (see the excellent survey of Lin and Gottschalk [21]). Most previous work deals with polygonal meshes and spline surfaces. A complete treatment for subdivision surfaces has not appeared in the literature. Unfortunately, subdivision surface interference tests cannot be based in a straightforward way on polygonal interference algorithms run on the control mesh (see Figure 2).

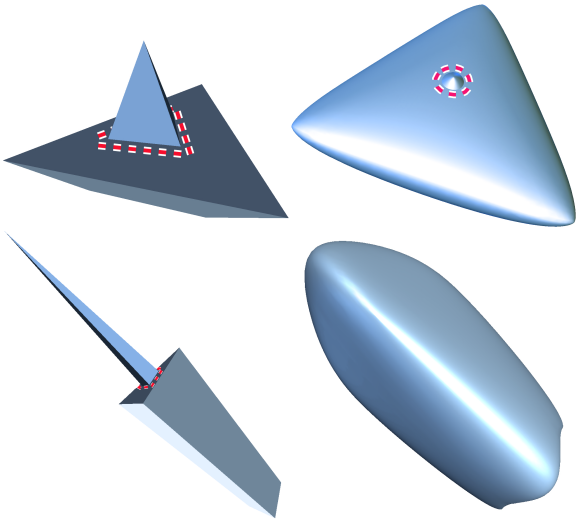

Figure 2: Control mesh check is insufficient. *Two self-intersecting control meshes (left) and (not shown at same scale) associated limit surfaces (right). A control mesh may be self-intersecting while the associated limit surface may (top) or may not (bottom) intersect.*

The collision detection problem is very difficult. Given N separate objects, there are $O(N^2)$ potential pair-wise collisions. Furthermore, if the objects are patches of a surface, then (typically) there are $O(N)$ pairs of adjacent objects whose shared seam should not be considered an interference region (see Figure 3). Most approaches employ a hierarchical bounding volume structure to avoid expensive intersection tests between well separated parts of the surface or multiple surfaces. Algorithms that do not specifically optimize self-interference queries perform poorly (if at all): they rely on spatial separation between non-interfering objects, and such separations do not exist between adjacent patches.

Our algorithm to detect (self-)interference for subdivision surfaces employs and builds on a number of previous ideas.

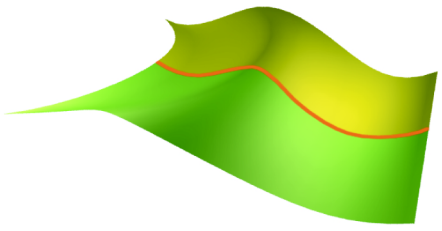

Figure 3: Self-interference check. *The detection algorithm should ignore the seam between patches.*

Collision Detection for Subdivision Surfaces was discussed by DeRose et al. [10] as part of a comprehensive treatment of character animation. As their approach is based on pointwise sampling of the subdivision surface at its control vertices, it may miss (self-)collisions. Also, they do not discuss optimizations specific to self-interference. In contrast, our approach will not miss (self-)collisions, and is particularly efficient for self-interference queries.

Collision Detection for Spline Surfaces Hughes et al. [17] describe an accurate algorithm for objects undergoing polynomial deformation. The algorithm uses linear programming, hierarchies, sweep-and-prune [7], and loop detection [15] to check for interference of B-spline (and Bézier) surface patches. The regular patches of a subdivision surface are splines, and many of these ideas could be applied if the surface deformation is polynomial.

Collision Detection for Polygonal Surfaces Volino and Thalmann present an algorithm for self-collision detection of polygonal surfaces [33]. They create a patch hierarchy, use surface curvature tests to rule out self-intersections, and use a combination of surface curvature and bounding box tests to rule out intersection between pairs of subsurfaces. Provot [26] confirms the soundness of this approach for polygons. Ponamgi et al. [25] present an algorithm for exact collision detection between non-convex polyhedral models. They describe a *hierarchical sweep and prune* strategy which allows them to exploit temporal coherence. Although we could incorporate this approach, we found that our patch hierarchy (Section 2.2 and Figure 5) effects similar pruning if each node in the hierarchy has a small number of children.

Interval Arithmetic has been used successfully for collision detection with many objects undergoing deformable motion in complex environments. Snyder et al. [29] described such a method, but it does not address self-collision nor subdivision surfaces. Extending this work to handle self-collision would involve a development akin to that presented in Section 3.1, in which we use interval analysis to bound normal variation over a surface patch.

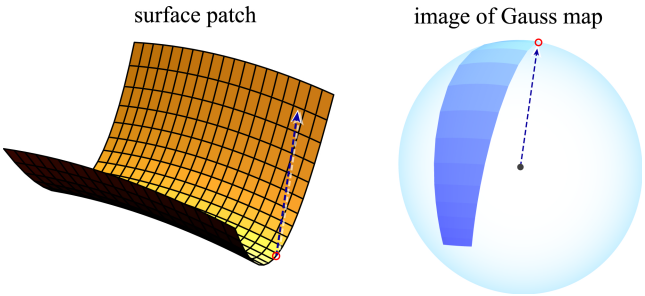

Figure 4: The Gauss map, $(u, v) \mapsto (\mathbf{f}_u \times \mathbf{f}_v)/\|\mathbf{f}_u \times \mathbf{f}_v\|_2$, *maps every point on the surface of an orientable 2-manifold to its unit normal vector [4]. An illustrative surface patch (left) and the image of its Gauss map on the unit sphere (right). Each point on the surface (arrow's tail, left image) has an associated surface normal (arrow, left image) that corresponds to a point on the unit sphere (arrow's head, right image).*

Surface Intersection Hohmeyer presents a general framework for robust surface-surface intersection based on loop detection [16]. His work covers quadric, parametric, as well as implicit surfaces. Krishnan and Manocha [20] treat algebraic and NURBS surfaces using a lower-dimensional formulation. Other notable work was published by Ma [24] and by Sederberg [27]. A commonality amongst all of these approaches is their use of bounds for the Gauss map of a surface patch (see Figure 4). We derive an interval bound

for the Gauss map of a subdivision surface, allowing us to leverage much of the collision detection work based on such bounds.

Our approach extends the framework presented by Thalmann and Volino [33], which effectively treats self-interference detection of arbitrary topology polygonal meshes. There are several possible ways to extend this framework to smooth subdivision surfaces. The naïve approach, detecting interference of the surface control mesh, leads to erroneous results (see Figure 2). An approximate approach, for applications that tolerate missed or false collisions, is to first discretize the limit surface into a polygonal mesh and then detect interference of the mesh. With this approach, our novel Gauss map bound can be used to guide the discretization and to bound the discretization error. The more precise approach we adopt applies the interference detection problem to the smooth limit surface, as in prior treatment of spline patches. We retain the general structure presented in [33] but instead of working with polygonal meshes we treat smooth surfaces of arbitrary topology.

2 Algorithm

In this section we describe the interference detection problem and present our solution.

The limit surface of subdivision is trivially parameterized over the control mesh triangles, implying a decomposition of the surface into coarse triangular *limit patches*. Since a refined control mesh describes the same limit surface, the decomposition of the surface may consist of arbitrarily fine triangular limit patches. We use the terms *patch* and *subpatch* (in contrast with *limit patch*) to denote a connected group of limit patches.

2.1 The Collision Detection Problem

The goal of the algorithm is to identify all self-intersecting limit patches, as well as all pairs of intersecting limit patches. The user provides a tolerance oracle that decides whether a pair of intersecting limit patches should be subdivided into finer limit patches in order to isolate the interfering regions more precisely (see Figures 1, 11, and 12).

2.2 Overview of Our Algorithm

We represent the limit surface at different resolutions via a hierarchy of patches (see Figure 5):

1. every level of the hierarchy represents the entire surface as a disjoint set of patches,
2. each patch is a disjoint union of its subpatches,
3. the root of the hierarchy is a single patch comprising the entire surface,
4. the leaves are limit patches, and
5. conceptually, every leaf patch has an infinite hierarchy of descendant limit patches, corresponding to the hierarchical parameterization induced by recursive subdivision of the control mesh.

See Garland et al. [12] for an excellent treatment of such hierarchies. To detect interference, we begin at the root patch of the hierarchy, and examine progressively finer subpatches at higher indexed levels. If we are able to guarantee that a patch does not self-intersect, we prune the search—we do not consider descendants of that patch. The detection algorithm has two significant routines:

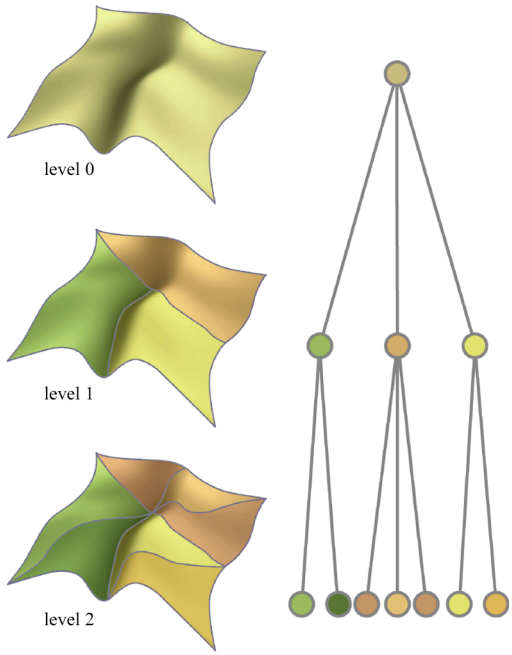

Figure 5: Hierarchy. *Illustrative example of a patch hierarchy whose leaves are limit patches. Note that level numbers increase towards finer levels and that the coarsest level of any surface consist of a single patch, i.e., the union of all limit patches.*

DetectPairwiseInterference accepts a pair of input patches, and returns a list of interfering regions between the patches. An interference region is represented as a pair of limit patches, where each input patch contributes one subpatch to the pair. This routine may make recursive calls to itself.

DetectSelfInterference accepts a single patch as input, and returns a list of interfering regions within the patch. An interference region is represented by a pair of limit patches belonging to the input patch. This routine may make recursive calls to itself, and it may call DetectPairwiseInterference.

2.3 Discussion of Pseudo-Code

We now turn to the definition of the detection algorithm given in Figure 6. In this discussion, $\mathcal{SP}(a)$ denotes the immediate subpatches of patch a.

DetectPairwiseInterference accepts a pair of patches $\{a, b\}$ and returns an interference list, each element an interfering pair $\{c_a, c_b\}$ of limit patches with $c_a \in \mathcal{SP}(a)$ and $c_b \in \mathcal{SP}(b)$. The algorithm is recursive: first, we attempt to rule out interference between a and b (Section 2.4); if interference cannot be ruled out, we check if both patches are sufficiently flat to be approximated as polygons. If so, interference is checked between the polygonal approximations. Otherwise we subdivide one of the patches, say b, into its constituent subpatches, and recursively check for interference between a and every subpatch of b.

DetectSelfInterference accepts a patch a and returns an interference list containing interfering pairs $\{c_1, c_2\}$ of limit patches with $c_1, c_2 \in \mathcal{SP}(a)$. The algorithm is recursive: first, we attempt to rule out the possibility of self-interference in patch a (Section 2.4). If self-interference cannot be ruled out, we subdivide patch a into its constituent subpatches $\mathcal{SP}(a)$, find all self-interferences within each subpatch via a recursive call to DetectSelfInterference, and find all interferences between pairs of subpatches through a call to DetectPairwiseInterference.

```
algorithm DetectSelfInterference(a)
  if RuleOutSelfInterference(a) return: ∅
  else return:
```

$$\bigcup_{c \in \mathcal{SP}(a)} \text{DetectSelfInterference}(c)$$

$$\cup$$

$$\bigcup_{c_1, c_2 \in \mathcal{SP}(a)} \text{DetectPairwiseInterference}(c_1, c_2)$$

```
// Assume level(b) ≥ level(a).
algorithm DetectPairwiseInterference(a,b)
  if RuleOutPairwiseInterference(a,b) return: ∅

  else if a and b sufficiently flat
    if poly. approxs. of a and b interfere
      return: {a, b}
    else return: ∅
  else if a not sufficiently flat
    return:
```

$$\bigcup_{c_a \in \mathcal{SP}(a)} \text{DetectPairwiseInterference}(b, c_a)$$

```
  else // b not sufficiently flat
      return:
```

$$\bigcup_{c_b \in \mathcal{SP}(b)} \text{DetectPairwiseInterference}(a, c_b)$$

Figure 6: *Pseudo-code for interference-detection algorithm.*

2.4 Details

The routines described above use bounds on the volume of a patch ("spatial bounds") and the Gauss map of a patch ("normal bounds") when they attempt to rule out interference between non-adjacent and adjacent patches respectively. Let us examine each test in more detail:

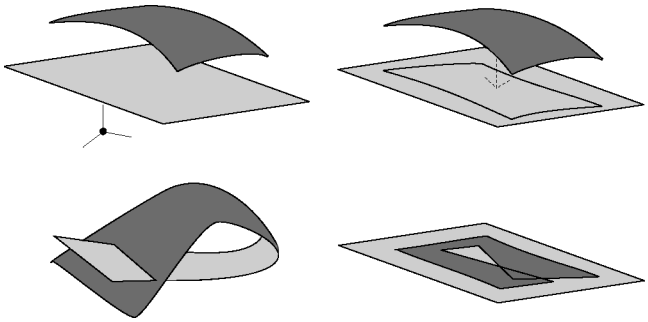

Figure 7: Self-interference test. *Sufficient conditions that a patch does not self-intersect are the existence of a separating plane between the origin and the image of Gauss map of the patch (top left), and the absence of interference in the projection of the patch contour onto the separating plane (top right). The intuition behind these criteria is as follows. The first ensures that the patch does not self-interfere due to bending (bottom left), while the second ensures that the patch does not self-interfere due to planar stretching and shearing (bottom right).*

The test to rule out interference between two patches depends on whether or not the patches are adjacent on the limit surface (we present an efficient adjacency test below). If the patches are not adjacent, then spatial bounds such as axis-aligned bounding boxes (AABBs) or oriented bounding boxes (OBBs) are used to rule out interference [7, 13]. The bounding box for a limit patch is built by bounding its control mesh, taking advantage of the local convex hull property. Note that repeated refinement of the control mesh of a patch yields progressively tighter spatial bounds. However subdivision is costly. In practice three steps of refinement are a good compromise for AABBs. The bounding boxes of the remaining patches are built by bounding the union of bounding boxes of their subpatches.

The test to determine if two patches are adjacent is defined as follows: given two patches *a* and *b*, with $\text{level}(a) \geq \text{level}(b)$, then the patches are adjacent *iff b* is an ancestor of some neighbor of *a*. This reflects the notion that patches are adjacent *iff* they are connected, regardless of the location of each patch in the hierarchy. To make this efficient the data structure representing a patch includes a list of all its immediate neighbors at the same level.

The test to rule out interference between two adjacent patches follows from the observation that the union of the two patches is itself a proper connected patch. Simply put, we apply the self-interference test to the union of the two patches.

The test to rule out self-interference is similar to the test given by Volino [33]. Two conditions must hold for ruling out self-intersection. First, there must exist a plane separating the image of the Gauss map from the origin (see Figure 7, top left). Secondly, the projection of the patch boundary onto this plane must lie in the plane and have no self-intersections (Figure 7, top right). For many applications it is reasonable to ignore the second condition; this has been independently noted by Provot [26] and Thalmann [8, 33], and we have confirmed this in our application to thin-shell simulation.

We now consider the check for a separating plane in more detail.

3 Self-Interference

The self-interference test requires that two conditions be met. Sections 3.1 discusses the first condition (Figure 7, top left) and Section 3.6 the second condition (Figure 7, top right). We shall focus on Loop's scheme [23]. All the results extend to the Catmull-Clark scheme and other popular schemes. In particular, our results rely on diagonalizing the extended subdivision matrix. This is not possible for some schemes, such as the Doo-Sabin scheme [30], but the results are easily extended to handle matrices which have non-trivial Jordan blocks. We have used this extension to handle valence three vertices in Loop's scheme.

Our final computations require interval analysis. An *interval* $\mathsf{A} = [a, b]$ is a closed subset of $\mathbb{R}$ with the usual arithmetic operations on intervals [1]. In the following, interval quantities are typeset in Sans Serif (e.g., $\mathsf{F}(\mathsf{X})$, $\mathsf{\Phi}_i(\Omega)$), matrix and vector values in boldface (e.g., $\mathbf{f}(x)$, $\mathbf{C}$), and scalar values in lowercase Roman (e.g., $f(x)$, c_i, $\phi_i(v, w)$).

3.1 Gauss Map Image Restricted to Hemisphere

As shown in Figure 7, to rule out self-interference we must prove the existence of a separating plane between the origin and the image of the Gauss map of the patch. An equivalent condition is that the set of all oriented normal vectors of the surface is contained in a non-degenerate cone (i.e., a cone with half-angle less than $\pi/2$). To determine if this cone exists, we derive analytic bounds on the normal vectors of a limit patch. The bounds we present are guaranteed to converge during subdivision.

When a patch has three regular corners (for Loop's scheme, these are vertices of valence six), we refer to the patch as *regular*; otherwise we refer to it as *irregular*. Regular patches define quartic box spline surfaces, and previous work has established effective approaches for bounding their Gauss map [16]. The literature does not address irregular patches; that is our focus.

To simplify the problem (without loss of generality), we shall: (a) consider only a single limit patch, (b) assume that at most one corner of any patch is an irregular vertex, and (c) apply a rigid-body transformation to the patch such that the irregular corner lies at the origin and the associated surface normal is collinear with the z-axis. Patches, i.e., groups of limit patches, have their Gauss map bounded by the union of the bounds over their constituent limit patches. The condition that a limit patch may have at most one irregular corner is easily satisfied by a single global subdivision step.

We begin by formulating expressions for tangent vectors of the irregular patch. In Section 3.3 we proceed to bound the tangent directions and then the normal directions of the patch.

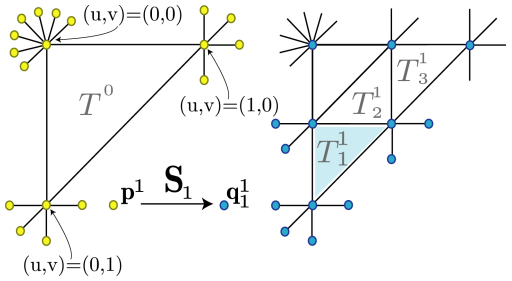

Figure 8: Loop subdivision. *One step of local subdivision around the irregular patch. On the left the parametric layout of an irregular patch. Applying the subdivision matrix $\mathbf{S}_1$, for example, produces the control points of the subpatch T_1^1 (shaded triangle).*

3.2 Scaled Tangent Vectors

Consider one step of subdivision around the irregular control triangle, as shown in Figure 8. Quadrisection produces three regular triangles and one irregular triangle. Let $\mathbf{p}$ be the $N \times 3$ vector of 3D control points at the coarsest level around the irregular triangle (Figure 8 top left), and let $\mathbf{p}^1$ be the $N \times 3$ vector of 3D control points in a similar neighborhood at the next level. Let $\mathbf{S}$ be the $N \times N$ subdivision matrix, such that $\mathbf{p}^1 = \mathbf{S}\mathbf{p}$. Similarly define $\mathbf{S}_1$ (respectively $\mathbf{S}_2$ and $\mathbf{S}_3$) as a rectangular subdivision matrix that relates $\mathbf{p}$ to the vector of control points $\mathbf{q}_1^1$ (respectively $\mathbf{q}_2^1$ and $\mathbf{q}_3^1$) around triangle T_1^1 (respectively T_2^1 and T_3^1). Note that after one step of subdivision, three quarters of the limit patch are defined by quartic box splines.

Our derivation is based on observing the eigenstructure of the subdivision matrix; this idea can be traced back to Doo and Sabin [11]. Assume that $\mathbf{S}$ is non-defective (if this assumption is relaxed a similar argument follows), and decompose the control vector $\mathbf{p}$ with respect to the eigenbasis $\{\mathbf{x}^n\}$, $n = 0 \,..\, N-1$, of $\mathbf{S}$:

$$\mathbf{p} = \mathbf{x}^0\mathbf{c}_0 + \mathbf{x}^1\mathbf{c}_1 + \mathbf{x}^2\mathbf{c}_2 + \dots ,$$

where $\mathbf{x}^n$ is an $N \times 1$ column eigenvector and $\mathbf{c}_i$ is a 1×3 vector. We write the $\mathbf{x}^i$ in order of non-increasing magnitudes of the eigenvalues λ_i. For simplicity, we further assume that $\lambda_1 = \lambda_2 = \lambda$ and $\lambda_0 > \lambda > |\lambda_3|$.

Suppose that the projection into the eigenbasis results in exactly one non-zero coefficient $\mathbf{c}_i$, so that $\mathbf{p} = \mathbf{x}^i\mathbf{c}_i$. One subdivision step produces the regular triangle T_1^1 surrounded by control points $\mathbf{q}_1^1 = \mathbf{S}_1\mathbf{p} = \mathbf{S}_1\mathbf{x}^i\mathbf{c}_i$. Since $\mathbf{x}^i$ is an eigenvector of $\mathbf{S}$, each step of subdivision scales the control points by the eigenvalue: the j^{th} step of subdivision generates the control points around T_1^j given by

$$\mathbf{q}_1^j = \mathbf{S}_1\mathbf{S}^{j-1}\mathbf{p} = \mathbf{S}_1\lambda_i^{j-1}\mathbf{x}^i\mathbf{c}_i = \lambda_i^{j-1}\mathbf{q}_1^1. \qquad (1)$$

Recall that for a regular patch the limit surface is a box spline, hence for the patch defined by $\mathbf{q}_1^1$ we can express the limit surface as $\mathbf{s}(u,v)|_{T_1^1} = \mathbf{B}(2u, 2v-1)\mathbf{q}_1^1$, where $\mathbf{B}(u,v)$ is a row vector of box spline basis polynomials defined over the unit triangle T_0. Furthermore, using the scaling relation given above, we can express the limit surface of the patch defined by $\mathbf{q}_1^j$ as $\mathbf{s}(u,v)|_{T_1^j} = \lambda_i^{j-1}\mathbf{B}(2^j u - 1, 2^j v)\mathbf{q}_1^1$.

The u-tangent of patch $\mathbf{s}(u,v)|_{T_1^j}$, defined as the vector function $\frac{\partial}{\partial u}\mathbf{s}(u,v)|_{T_1^j}$, describes its tangents in the u parameter direction. Recall that we want to bound the *direction* of the tangent vectors, and not the magnitude. We multiply the tangent vector by the positive scalar $(2\lambda)^{1-j}$, noting that this does not alter its direction. The resulting scaled tangent vector is

$$\begin{aligned} \mathbf{h}_\mathbf{u}(u,v)\Big|_{T_1^j} &= (2\lambda)^{1-j}\frac{\partial}{\partial u}\mathbf{s}(u,v) \\ &= \underbrace{2\,(\lambda_i/\lambda)^{j-1}\,\mathbf{B}_u(2^j u - 1, 2^j v)\mathbf{S}_1\mathbf{x}^i}_{\phi_u^i(u,v)\Big|_{T_1^j}}\mathbf{c}_i \qquad (2) \end{aligned}$$

where $\mathbf{B}_u(u,v) = \frac{\partial}{\partial u}\mathbf{B}(u,v)$, and $\phi_u^i(u,v) : \mathbb{R}^2 \mapsto \mathbb{R}$ is a partial derivative of the *scaled eigenbasis function* corresponding to the eigenvector $\mathbf{x}^i$ (with its restriction over T_1^j defined above). $\phi_u^i(u,v)$ is also defined over T_2^j and T_3^j (by an analogous derivation), thus it is well-defined over the entire domain T_0. Consequently, the scaled u-tangent has the simple expression $\mathbf{h}_\mathbf{u}(u,v) = \mathbf{c}_i\phi_u^i(u,v)$. A similar derivation establishes the expression for the scaled v-tangent, $\mathbf{h}_\mathbf{v}(u,v) = \mathbf{c}_i\phi_v^i(u,v)$.

Up to this point, we have assumed that the expansion of the control vector $\mathbf{p}$ in the eigenbasis of $\mathbf{S}$ is $\mathbf{p} = \mathbf{x}^i\mathbf{c}_i$ (with fixed i). We now lift this restriction, and treat the case of the general expansion $\mathbf{p} = \sum_{i=0}^{N-1}\mathbf{x}^i\mathbf{c}_i$. Since the subdivision operator S, the differentiation operator $\frac{\partial}{\partial u}$, and multiplication by a scalar $(2\lambda)^{1-j}$ are all linear operators, the scaled tangents are given by

$$\mathbf{h}_\mathbf{u}(u,v) = \sum_{i=1}^{N-1}\mathbf{c}_i\phi_u^i(u,v) \qquad \mathbf{h}_\mathbf{v}(u,v) = \sum_{i=1}^{N-1}\mathbf{c}_i\phi_v^i(u,v).$$

The terms corresponding to $i = 0$ have been dropped because they are always zero.

3.3 Analytic Bound of Surface Normals

We want to bound the scaled normal, $\mathbf{h}_\mathbf{u}(u,v) \times \mathbf{h}_\mathbf{v}(u,v)$. The desired quantity $\mathsf{h}_\mathsf{u} \times \mathsf{h}_\mathsf{v}$ is a 3D vector with scalar interval components, and it can be interpreted geometrically as an axis-aligned bounding box around the heads of the scaled normal vectors. We would like the bound to be tight if we apply it to a sufficiently fine limit triangle; if the bound is not sufficiently tight, we shall subdivide the limit triangle and apply the bound separately to each sub-triangle. The shape of a sufficiently fine limit triangle with eigen-coefficients $\mathbf{c}_i$ is governed by the subdominant eigenterms of its eigenbasis expansion. Hence it is critical to find an expression for $\mathsf{h}_\mathsf{u} \times \mathsf{h}_\mathsf{v}$ that is tight when $\{\mathbf{c}_1, \mathbf{c}_2\} \gg |\mathbf{c}_i|$, $i > 2$.

Let us rewrite $\mathbf{h}_\mathbf{u}(u,v) = \mathbf{h}_\mathbf{u}^{s.d.}(u,v) + \mathbf{h}_\mathbf{u}^{h.o.t.}(u,v)$ as the sum of the subdominant terms $\mathbf{h}_\mathbf{u}^{s.d.}(u,v) = \mathbf{c}_1\phi_u^1(u,v) + \mathbf{c}_2\phi_u^2(u,v)$ and higher order terms $\mathbf{h}_\mathbf{u}^{h.o.t.}(u,v) = \sum_{i=3}^{N-1}\mathbf{c}_i\phi_u^i(u,v)$. Similarly we rewrite, $\mathbf{h}_\mathbf{v}(u,v) = \mathbf{h}_\mathbf{v}^{s.d.}(u,v) + \mathbf{h}_\mathbf{v}^{h.o.t.}(u,v)$. By the

distributive property of the cross product, the scaled normal is the sum of four terms, which we shall bound individually and then add using interval arithmetic:

$$\begin{aligned}\mathsf{h_u} \times \mathsf{h_v} \quad = \quad & (\mathsf{h_u^{s.d.}} \times \mathsf{h_v^{s.d.}}) + (\mathsf{h_u^{s.d.}} \times \mathsf{h_v^{h.o.t.}}) + \\ & (\mathsf{h_u^{h.o.t}} \times \mathsf{h_v^{s.d.}}) + (\mathsf{h_u^{h.o.t}} \times \mathsf{h_v^{h.o.t.}}). \end{aligned} \tag{3}$$

The first term is a bound over the function $\mathbf{h}_\mathbf{u}^{s.d.}(u,v) \times \mathbf{h}_\mathbf{v}^{s.d.}(u,v) = \phi_{1\times 2}(u,v)(\mathbf{c}_1 \times \mathbf{c}_2)$, where $\phi_{1\times 2}(u,v) = \phi_u^1(u,v)\phi_v^2(u,v) - \phi_u^2(u,v)\phi_v^1(u,v)$. We precompute the scalar interval bound $\Phi_{1\times 2}$ on $\phi_{1\times 2}(u,v)$ over T^0, and at runtime evaluate $\mathsf{h_u^{s.d.}} \times \mathsf{h_v^{s.d.}} = \Phi_{1\times 2}(\mathbf{c}_1 \times \mathbf{c}_2)$.

To compute the remaining terms we compute interval bounds for the intermediate factors:

$$\begin{array}{ll} \mathsf{h_u^{s.d.}} = \mathbf{c}_1\Phi_\mathsf{u}^1 + \mathbf{c}_2\Phi_\mathsf{u}^2 & \mathsf{h_v^{s.d.}} = \mathbf{c}_1\Phi_\mathsf{v}^1 + \mathbf{c}_2\Phi_\mathsf{v}^2 \\ \mathsf{h_u^{h.o.t.}} = \sum_{i=3}^{N-1} \mathbf{c}_i\Phi_\mathsf{u}^\mathsf{i} & \mathsf{h_v^{h.o.t.}} = \sum_{i=3}^{N-1} \mathbf{c}_i\Phi_\mathsf{v}^\mathsf{i}, \end{array}$$

where $\Phi_\mathsf{u}^\mathsf{i}$ and $\Phi_\mathsf{v}^\mathsf{i}$ are precomputed interval bounds on $\phi_u^i(u,v)$ and $\phi_v^i(u,v)$ over T^0 (we shall develop these bounds in Section 3.4). Taking interval cross products of the above factors the three remaining terms are formed.

The proof of convergence follows from the observation that $|\mathbf{a} \times \mathbf{b}| \leq |\mathbf{a}||\mathbf{b}|$, and hence the three terms in Equation 3 that involve $\mathsf{h_u^{h.o.t.}}$ and $\mathsf{h_v^{h.o.t.}}$ vanish during subdivision. The remaining term, $\Phi_{1\times 2}(\mathbf{c}_1 \times \mathbf{c}_2)$, is the product of a scalar interval and a vector almost parallel to the z-axis. If the scalar interval $\Phi_{1\times 2}$ does not include some finite neighborhood around zero then interval arithmetic will produce a tight bound on the (z-axis) direction, $\mathbf{h_u}(u,v) \times \mathbf{h_v}(u,v)$.

3.4 Analytic Bound on Partial Derivatives of Scaled Eigenbasis Functions

In this section we describe how to precompute the scalar intervals $\Phi_\mathsf{u}^\mathsf{i}$ and $\Phi_\mathsf{v}^\mathsf{i}$, which bound $\phi_u^i(u,v)$ and $\phi_v^i(u,v)$ over T_0. These functions depend on the eigenvectors of $\mathbf{S}$ (and hence on the valence of the irregular corner vertex), but since they are independent of the control vector $\mathbf{p}$, these intervals are precomputed.

It is easy to bound $\phi_u^i(u,v)$ over the domain $T^1 = T_1^1 \cup T_2^1 \cup T_3^1$, since the scalar function is a polynomial over each of the three subdomains. Elementary Calculus yields the analytic bound $\Phi_\mathsf{u}^\mathsf{i}|_{T^1} = \text{range}\left[\phi_u^i(u,v),\ (u,v) \in T^1\right]$. To extend this bound over the entire domain T_0, we observe a scaling relation on $\phi_u^i(u,v)$ that is evident from Equation 2:

$$\begin{aligned}&\text{range}\left[\phi_u^i(u,v),\ (u,v) \in T^j\right] = \\ &\qquad (\lambda_i/\lambda)^{j-1}\,\text{range}\left[\phi_u^i(u,v),\ (u,v) \in T^1\right], \end{aligned} \tag{4}$$

whence it follows directly that the bound over the entire domain T_0 is the union of an infinite sequence of bounds over the domains T^j:

$$\Phi_\mathsf{u}^\mathsf{i} = \bigcup_{j=1}^{\infty} \Phi_\mathsf{u}^\mathsf{i}|_{T^j} = \bigcup_{j=1}^{\infty} (\lambda_i/\lambda)^{j-1}\,\Phi_\mathsf{u}^\mathsf{i}|_{T^1}\ . \tag{5}$$

Let us examine how this expression can be simplified. We examine three cases: $i = 0$, $i \in \{1,2\}$, and $i > 2$.

- $i = 0$ We assume that $\mathbf{x}^0 = [1, 1 \ldots 1]^T$, as this is required for the subdivision surface to be affine invariant [34]. Then by the partition of unity property of the box spline basis, $\mathbf{B}\mathbf{x}^0$ is unity and $\mathbf{B}_u\mathbf{x}^0$ is zero, and from Equation 2 we conclude that $\phi_u^0(u,v)$ is zero everywhere and $\Phi_\mathsf{u}^0 = [0,0]$.
- $i \in \{1,2\}$ Since $\lambda_1 = \lambda_2 = \lambda$, the geometric scaling factor λ_i/λ is unity. In this case, our simplified expression is $\Phi_\mathsf{u}^1 = \Phi_\mathsf{u}^1|_{T^1}$, i.e., the bound over T^1 is a bound over T_0.
- $i > 2$ In this case, the geometric scaling factor λ_i/λ is less than unity, hence the sequence of intervals $(\lambda_i/\lambda)^{j-1}\,\Phi_\mathsf{u}^\mathsf{i}|_{T^1}$ vanishes as $j \to \infty$ and the interval $\Phi_\mathsf{u}^\mathsf{i} = \left[\min\left(0, \Phi_\mathsf{u}^1|_{T^1}\right), \max\left(0, \Phi_\mathsf{u}^1|_{T^1}\right)\right]$ is a conservative bound on $\phi_u^i(u,v)$ over T_0. Simply put, we compute a bound over T^1, and extend that bound to include zero.

Recall that in Equation 2 we scaled the tangents by $(2\lambda)^{1-j}$. We now see the effect: the scaling factor (in Equation 5) is $|\lambda_i/\lambda| \leq 1$, $i > 0$; consequently the infinite geometric sequence converges. If the tangents were not scaled, the scaling factor would be $2\lambda_i > 1$ for some valences and the union over the infinite sequence would not converge. The second consequence of scaling the tangent vectors is that the scaling factor is now unity for $i \in \{1,2\}$, hence the bound on the subdominant eigenbasis functions is tight.

The scalar interval bound $\Phi_{1\times 2}$ on $\phi_{1\times 2}(u,v)$ over T^0 is computed in the same manner. It is easy to bound $\phi_{1\times 2}(u,v)$ over the domain T^1, since the scalar function is a polynomial over each of the three subdomains. Elementary Calculus yields an analytic bound, and since all the component eigenbasis functions are associated with subdominant eigenvalues, the function $\phi_{1\times 2}(u,v)$ obeys a scaling relation analogous to Equation 4, with the scaling factor unity. Hence the bound on $\phi_{1\times 2}(u,v)$ over the domain T^1 serves as the bound $\Phi_{1\times 2}$ over the domain T^0.

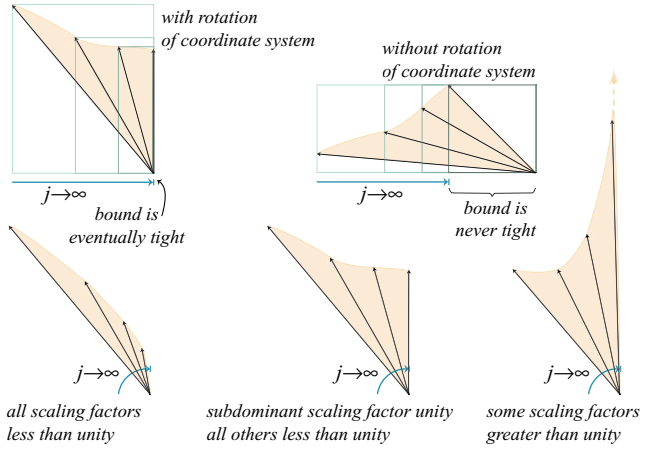

Figure 9: Tight bounds. Top row: *Axis-aligned bounding boxes provide tight bounds for axis-aligned directions, favoring rotation of the coordinate system.* Bottom row: *We scale the tangent vectors by $(2\lambda)^{1-j}$, which results in normalized scaling factors $|\lambda_i/\lambda|$ for the eigenfunctions; consequently the tangent vectors are well-behaved near the irregular vertex (middle). If we omit this scaling, the tangent vectors may be poorly behaved (left and right).*

3.5 Obtaining Tighter Bounds

We noted previously that $\mathsf{h_u}$ and $\mathsf{h_v}$ can be interpreted as axis-aligned bounding boxes (AABBs) around heads of the tangent vectors. Let us examine the consequence of using an AABB as a bounding volume. If we want to express a bound over a single point, or an axis-aligned line segment, the bounding box is tight. However, if we want to express a bound over an arbitrary line segment, an AABB may have significant slack and present a poor choice.

It is important for us to have a tight bound on the tangents (and normals) when the patch is flat. Note that when the patch is flat, the heads of all the normals vectors will lie along some line segment. An AABB will give a tight bound on the normal direction only if this line segment is aligned with an axis. This explains why we

rotate the control mesh so that the normal vector associated with the irregular vertex is aligned with the z-axis. The detection algorithm examines progressively smaller neighborhoods around the irregular patch; eventually all the normal vectors in the neighborhood will be sufficiently aligned with the z-axis.

3.6 Self-Interference of Projected Contour

We now turn our attention to the second condition in the interference test: The projection of the patch contour onto the separating plane is a curve, and it must not self-interfere. In essence, this self-intersection problem for curves is a lower-dimensional version of the self-intersection problem for patches.

Consider applying a rigid-body transformation to the separating plane and the patch such that the separating plane is the xy-plane. A projection onto the separating plane amounts to setting the z-coordinate to zero. Since the subdivision operator affects each coordinate independently, we note that the projected contour is a subdivision curve defined by the projection of part of the control mesh onto the plane. Specifically, only the control triangles in the two-ring of the contour are required to define the limit contour. Self-interference detection for the contour curve is simply a lower-dimensional instance of the problem addressed in this paper.

One simplification that is possible in the curve setting is that instead of bounding normal directions, we may bound tangent directions, since for curves the normal and tangent directions are related by a rotation through $\pi/2$. Also note that in this two-dimensional setting, the self-interference test analogous to that described in Section 2 has only one condition: the tangent directions must be bounded by a wedge with half-angle less than $\pi/2$.

4 Results

We have implemented the algorithm described in the previous section as part of a larger system for thin-shell simulation. We derived and then tabulated the bounds on the eigenfunction derivatives, Φ^i_u, Φ^i_v, and normals, $\Phi_{1\times 2}$, in a symbolic algebra package. At run time the bounds, together with the left eigenvectors needed to compute the eigencoefficients $\mathbf{c}_i$, are loaded. The bounds $h_u \times h_v$ are then computed as described in Section 3.3.

We ensure that the half-angle of our normal bounding cone is tight within $\pi/180$ (1 degree) of the exact infinite series of bounds over the regular subpatches. To achieve this, we subdivide a given patch by multiplying the eigencoefficients by the eigenvalues (without actually altering the mesh data structure), extracting the three regular subpatches (by multiplying the eigencoefficients by the right eigenvectors and using picking matrices), and finally computing a normal bound over the regular subpatches. If the half-angle of the normal bounding cone over the entire patch is not within $\pi/180$ of the corresponding half-angle over the three regular subpatches, we recursively bound the normals of the irregular subpatch until the bound is tight enough; at this point, we have conceptually subdivided the original patch into a set consisting of regular subpatches and a single small irregular subpatch (without altering the mesh data structure). The bound over the original patch is computed as the union of the bounds over this set of subpatches.

Since the bound over regular subpatches is based on known methods for splines, and the number of iterations is typically only four or five, our bounding method runs within a small constant factor of the speed of methods based on splines. In practice only a small portion of the total detection time is spent in functions associated with irregular limit patch Gauss map bounds. Consequently there is very little runtime difference between a spline patch interference detection code and our more general subdivision surface interference detection code. Our prototype implementation uses rudimentary bounding primitives: spatial bounds are represented as axis-aligned bounding boxes, and normal bounds are represented by normal cones [18]. Researchers have established that the performance of collision detection algorithms benefits greatly from using tighter spatial and normal bounds. In particular we believe that our approach would benefit from using ShellTrees [19] or QuOSPO trees [14] as spatial bounds, since they have superior convergence properties compared to AABBs. Similarly we would benefit from using convex pyramids [9, 28] to represent normal bounds. For the examples we present in this paper, the collision detection time ranges from 0.1 s to 30 s.

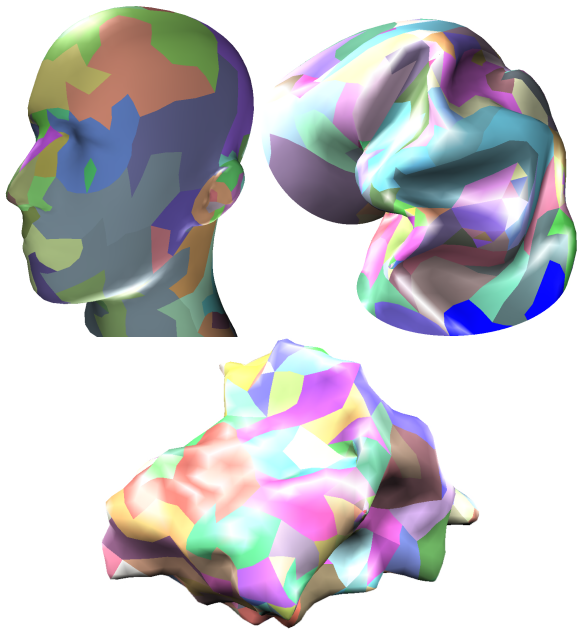

Figure 10: Experimental verification of self-interference test. *In relatively flat regions large patches, i.e., groups of limit patches, are determined to be without self intersection based on the Gauss map criterion. Each such group is indicated with a single color. Note that the groups are smaller in extent near high curvature areas.*

Figure 10 demonstrates our algorithm's ability to rule out self-interference. The coloring of the mannequin, lyase molecule energy field, and bent tube was generated by traversing down the patch hierarchy, and pruning the traversal at those patches for which we ruled out self-interference; the colors delineate those patches of the hierarchy that did not require further subdivision. Note that in flat regions of the surface (such as the base and top of the tube, and the forehead of the mannequin) the extent of the patches is relatively large, whereas in regions with high curvature (such as the nose and ear of the mannequin, and more notably some peaks in the lyase model), the extent of the patches is rather small. These examples demonstrate the dependence of our self-interference criterion on the curvature of the surface patch.

We have used our algorithm to find self-interfering regions of various models, ranging from geometric curiosities to physically simulated objects. Figures 1, 11, and 12 show the interference regions we found for three models, with progressively higher precision (recall that the user may provide an oracle that determines whether a pair of interfering limit patches should be subdivided further to increase precision). The bent tube model was generated using a thin-shell physical simulator [5]. The other two models were created specifically to illustrate self-interference of closed surfaces. The multi-handled torus illustrates complex topology while the partially everted sphere consists mostly of regular patches and simple topology. Given equal implementation effort our runtime for the latter kind of example will be close to that for a corresponding spline model.

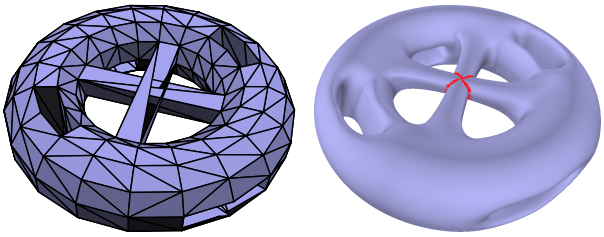

Figure 11: Self-interference detection. *Control mesh and interference resolved limit surface of higher genus torus with two tubes crossing in the center.*

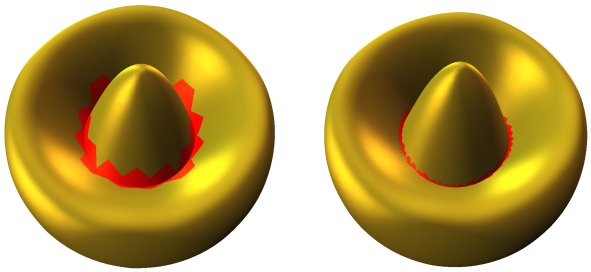

Figure 12: User-controlled tolerance. *A partially everted sphere and its interference region resolved to different resolutions.*

5 Conclusion

We have derived an interval bound on the normal directions of a subdivision surface near an irregular vertex using eigenbasis functions induced by the local subdivision operator. The bounds have been found to work well in practice, i.e., they are tight and require only a small number of multiplications in the eigenbasis—equivalent to standard subdivision, but without any change to the mesh data structures.

Such normal bounds have many applications since they are a central element in a large number of algorithms related to interference detection [3, 5, 10, 16, 18, 22]. We demonstrated an example application of these bounds to self-collision detection. In future work we hope to include more optimizations for other parts of the collision detection algorithm [9, 14, 19, 28] and pursue stable collision response methods in the context of thin-shell modeling.

Acknowledgements The research reported here was supported in part through NSF (DMS-9874082, DMS-9872890, ACI-9982273), the Packard Foundation, Pixar, Alias|Wavefront, Intel, and Lucent. Special thanks to Adi Levin, Mathieu Desbrun, Mika Nyström, Nathan Litke, Denis Zorin, Jos Stam, Jeff Bolz, Zoë Wood, Yuval Grinspun and Khrysaundt Koenig.

References

[1] ALEFELD, G., AND HERZBERGER, J. *Introduction to Interval Computations*. Academic Press, 1983.

[2] BARAFF, D., AND WITKIN, A. Large steps in cloth simulation. *Proceedings of SIGGRAPH 98* (July 1998), 43–54.

[3] BIERMANN, H., KRISTJANSSON, D., AND ZORIN, D. Approximate boolean operations on free-form solids. In *Proceedings of SIGGRAPH 2001* (Los Angeles, CA, August 2001). In Press.

[4] CARMO, M. P. D. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976, ch. 3.

[5] CIRAK, F., ORTIZ, M., AND SCHRÖDER, P. Subdivision surfaces: A new paradigm for thin-shell finite-element analysis. *Internat. J. Numer. Methods Engrg. 47* (2000), 2039–2072.

[6] CIRAK, F., SCOTT, M. J., ANTONSSON, E. K., ORTIZ, M., AND SCHR ÖDER, P. Integrated modeling, finite-element analysis, and engineering design for thin-shell structures using subdivision. *Computer-Aided Design* (2001). In Press.

[7] COHEN, J. D., LIN, M. C., MANOCHA, D., AND PONAMGI, M. K. I-collide: An interactive and exact collision detection system for large-scale environments. In *Proc. ACM Interactive 3D Graphics Conf.* (1995), pp. 189–196.

[8] COURSHESNES, M., VOLINO, P., AND THALMANN, N. M. Versatile and efficient techniques for simulating cloth and other deformable objects. *Proceedings of SIGGRAPH 95* (August 1995), 137–144.

[9] DANIEL, M. Using a convex pyramid to bound surface normal vectors. *Computer Graphics Forum 15*, 4 (1996), 219–227.

[10] DEROSE, T., KASS, M., AND TRUONG, T. Subdivision surfaces in character animation. *Proceedings of SIGGRAPH 98* (July 1998), 85–94.

[11] DOO, D., AND SABIN, M. Analysis of the behaviour of recursive division surfaces near extraordinary points. *Computer Aided Geometric Design 10*, 6 (1978), 356–360.

[12] GARLAND, M., WILLMOTT, A., AND HECKBERT, P. S. Hierarchical face clustering on polygonal surfaces. *2001 ACM Symposium on Interactive 3D Graphics* (March 2001), 49–58.

[13] GOTTSCHALK, S., LIN, M., AND MANOCHA, D. Obb-tree: A hierarchical structure for rapid interference detection. *Proceedings of SIGGRAPH 96* (August 1996), 171–180.

[14] HE, T. Fast collision detection using quospo trees. *1999 ACM Symposium on Interactive 3D Graphics* (April 1999), 55–62.

[15] HOHMEYER, M. A surface intersection algorithm based on loop detection. *International Journal of Computational Geometry and Applications 1*, 4 (1991), 473–490.

[16] HOHMEYER, M. E. *Robust and Efficient Intersection for Solid Modeling*. PhD thesis, UC Berkeley, 1992.

[17] HUGHES, M., DIMATTIA, C., LIN, M. C., AND MANOCHA, D. Efficient and accurate interference detection for polynomial deformation. *Proceedings of Computer Animation '96* (1996).

[18] KIM, D.-S., PAPALAMBROS, P. Y., AND WOO, T. C. Tangent, normal, and visibility cones on bézier surfaces. *Computer Aided Geometric Design 12*, 3 (1995), 305–320.

[19] KRISHNAN, S., GOPI, M., LIN, M., MANOCHA, D., AND PATTEKAR, A. Rapid and accurate contact determination between spline models using shell-trees. *Computer Graphics Forum 17*, 3 (1998), 315–326.

[20] KRISHNAN, S., AND MANOCHA, D. An efficient surface intersection algorithm based on lower-dimensional formulation. *ACM Transactions on Graphics 16*, 1 (January 1997), 74–106.

[21] LIN, M., AND GOTTSCHALK, S. Collision detection between geometric models: A survey. *Proceedings of IMA Conference on Mathematics of Surfaces* (1998).

[22] LITKE, N., LEVIN, A., AND SCHRÖDER, P. Trimming for subdivision surfaces. *Computer Aided Geometric Design 18*, 5 (June 2001), 463–481.

[23] LOOP, C. T. Smooth subdivision surfaces based on triangles. Master's thesis, Department of Mathematics, University of Utah, 1987.

[24] MA, Y., AND LEE, Y.-S. Detection of loops and singularities of surface intersections. *Computer-Aided Design 30*, 14 (1998), 1059–1067.

[25] PONAMGI, M. K., MANOCHA, D., AND LIN, M. C. Incremental algorithms for collision detection between polygonal models. *IEEE Transactions on Visualization and Computer Graphics 3*, 1 (Jan.–Mar. 1997), 51–64.

[26] PROVOT, X. Collision and self-collision handling in cloth model dedicated to design. *Computer Animation and Simulation '97* (September 1997), 177–190.

[27] SEDERBERG, T., AND MEYERS, R. Loop detection in surface patch intersections. *Computer Aided Geometric Design 5*, 2 (1988), 161–171.

[28] SEDERBERG, T. W., AND ZUNDEL, A. K. Pyramids that bound surface patches. *Graphical Models and Image Processing 58*, 1 (January 1996), 75–81.

[29] SNYDER, J. M., WOODBURY, A. R., FLEISCHER, K., CURRIN, B., AND BARR, A. H. Interval method for multi-point collision between time-dependent curved surfaces. *Proceedings of SIGGRAPH 93* (August 1993), 321–334.

[30] STAM, J. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. *Proceedings of SIGGRAPH 98* (July 1998), 395–404.

[31] TERZOPOULOS, D., PLATT, J., BARR, A., AND FLEISCHER, K. Elastically deformable models. *Computer Graphics (Proceedings of SIGGRAPH 87) 21*, 4 (July 1987), 205–214.

[32] TERZOPOULOS, D., PLATT, J., AND FLEISCHER, K. Heating and melting deformable models (from goop to glop). *Graphics Interface '89* (June 1989), 219–226.

[33] VOLINO, P., AND THALMANN, N. M. Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity. *Computer Graphics Forum 13*, 3 (1994), 155–166.

[34] ZORIN, D., AND SCHRÖDER, P., Eds. *Subdivision for Modeling and Animation*. Course Notes. ACM SIGGRAPH, 1998.