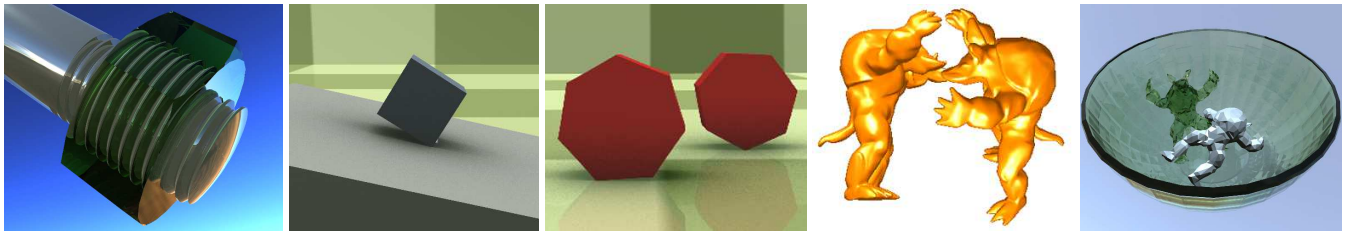
Adaptive Image-based Intersection Volume

Bin Wang*
Beihang University,
University of British Columbia

François Faure†
University of Grenoble, INRIA, LJK-CNRS,
University of British Columbia

Dinesh K. Pai‡
University of British Columbia



(a) (b) (c) (d) (e)

Figure 1: *Examples of challenging contact scenarios handled by our method.* (a) *The movement of a tight fitting nut on a bolt can be simulated directly using the geometric models.* (b) *Very small geometric features on a flat surface can dramatically change the behavior of objects sliding on it.* (c) *"Ruina wheels." Two almost identical wheels have obviously different rolling behavior due to subtle features (one is slightly convex and another is slightly concave); our method can simulate this contact behavior realistically.* (d) *A simulation with* 4.4 *million triangles.* (e) *A snapshot of an interactive simulation with ray-traced rendering.*

Abstract

A method for image-based contact detection and modeling, with guaranteed precision on the intersection volume, is presented. Unlike previous image-based methods, our method optimizes a non-uniform ray sampling resolution and allows precise control of the volume error. By cumulatively projecting all mesh edges into a generalized 2D texture, we construct a novel data structure, the Error Bound Polynomial Image (EBPI), which allows efficient computation of the maximum volume error as a function of ray density. Based on a precision criterion, EBPI pixels are subdivided or clustered. The rays are then cast in the projection direction according to the non-uniform resolution. The EBPI data, combined with ray-surface intersection points and normals, is also used to detect transient edges at surface intersections. This allows us to model intersection volumes at arbitrary resolution, while avoiding the geometric computation of mesh intersections. Moreover, the ray casting acceleration data structures can be reused for the generation of high quality images.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Geometric algorithms;

Keywords: Collision detection, gpu, ray casting.

Links: DL PDF

*e-mail: binwangbuaa@gmail.com
†e-mail:francois.faure@inria.fr
‡e-mail:pai@cs.ubc.ca

1 Introduction

The computation of polyhedron intersections is an important problem in Computer Graphics, with applications in collision detection and computer assisted design. Image-based approximation methods (reviewed in Section 2.2) have been proposed for applications where fast computation times are necessary, such as physical simulation [Allard et al. 2010] or interactive design [Hable and Rossignac 2005]. Tuning the resolution for physical simulation is more difficult than for visualization, since small details may have important consequences on the object trajectories. Since computation time and memory footprint strongly increase along with image resolution, the simulation of precise contact between complex models has remained out of the scope of image-based volume contact methods so far. We present a novel approach to optimize non-uniform image resolution with guaranteed precision on the intersection volume, applied to volume-based collision response.

Our method uses the error of the intersection volume as the refinement criterion. The volume is numerically integrated by sampling the surfaces using parallel rays, and multiplying the intersection depth with the area associated with each ray. We introduce a novel concept, the Error Bound Polynomial Image (EBPI), which allows us to first rasterize the mesh edges at moderate resolution, and then to compute the necessary sampling density to guarantee a given precision of the volume. Storing and processing of depth intervals on a per-ray basis, combined with geometry rasterization at moderate resolution, alleviate the memory limitations of previous image-based approaches. blue Moreover, intersection volumes are supported by not only mesh edges, but also transient edges created by surface intersections. We show how to maintain the precision criterion without computing the actual intersections edges.

We harness the increasing power of GPU ray bluetracing to implement adaptive sampling, and the acceleration structures are straightforwardly reused for physics-based realistic rendering. With the ever-growing geometrical complexity of the models, the need for high quality rendering, and the increasing performance of ray-tracers, we believe that the ability of leveraging ray tracing acceleration structures for collision detection will be increasingly relevant.

Our specific contributions are the following:

- the Error Bound Polynomial Image to compute polyhedron volumes with guaranteed precision and optimized sampling;
- the inference of intersection edges based on ray intersection points and EBPI;
- various accelerations such as the optimization of the sampling directions using an Extended Gaussian Image of the surfaces.

The remainder of this paper is organized as follows. Section 3 provides an overview of the entire algorithm in the simpler 2D setting, followed by the details of the algorithm in the full 3D setting in Section 4. Special treatment for precise intersection volume sampling is discussed in Section 5. Implementation and results are discussed in Section 6 and Section 7, and we finally conclude in Section 8.

2 Background

2.1 Related work

The idea of using parallel rays to detect interference between volume objects can be traced back to Menon et al. [1994]. Due to the high computational cost of ray tracers, most of the subsequent work on image-based collision detection leveraged the rasterization hardware, and after a decade of progress Heidelberger et al. [2004b] came up with an efficient detection of collision and self-collision between non-convex volume objects using Layered Depth Images *(LDIs)*. Baciu and Wong [2004] performed image-based triangle collision detection for cloth simulation by comparing triangle indices in neighboring pixels, using multiple renderings passes with sophisticated camera setups. Galloppo et al. [2006] locally projected object patches to textures on contact planes and detect collisions based on texel depth.

Collision detection is often the bottleneck in physically based animation, and the field is too large to be reviewed in detail here. We refer the reader to the survey of Teschner et al. [2004] on collision detection between deformable objects. In object space, Dingliana and O'Sullivan [2000] presented a time-critical approach for collision detection and response between sphere trees, Barbic and James [2007] proposed a method for the detection between a rigid and a reduced deformable body with complex geometries. Other authors leverage the increasing power of the GPU to detect collisions in object space, typically using continuous detection between triangles. Tang et al. [2011] developed a high performance framework on the GPU for rigid objects as well as deformable models with self-collision detection. A front tracking strategy [Tang et al. 2009] within a bounding volume hierarchy is used to exploit temporal coherency in the generation of pairs of potentially colliding primitives, then a series of high performance streamed computations are applied to maximize the parallelism while progressively computing pairs of actually colliding primitives. In general, self-collision tests are the most difficult to cull out, and can take up to 90% of the computation time. Govindaraju et al. [2005] propose a precomputed chromatic decomposition of the meshes. Barbic and James [2010] precompute culling criteria based on reduced coordinates to quickly cull out tests in case of moderate deformations. Schvartzman et al. [2010] recently improved the self-intersection culling tests.

When object intersection can not be avoided using continuous collision detection, the repulsion forces are typically computed based on penetration depth, which is complex. Sud et al. [2006] use the graphics hardware to accelerate the computation of distance fields. Hermann et al. [2008] use a ray-tracer in object space to robustly handle deep penetrations between volumetric objects. Je et al. [2012] use optimization techniques to efficiently compute penetration depth in object space. Penetration depth can also be ambiguous. Heidelberger et al.[2004a] propagate it from the surface of one object through the volumetric mesh of the other to avoid inconsistencies. Harmon et al. [2011] introduced space-time interference volumes *(STIV)* to extend the idea of volume contact to continuous collision detection. Although the precise computation of STIVs is difficult to perform efficiently, they have been successfully applied to interactive modeling.

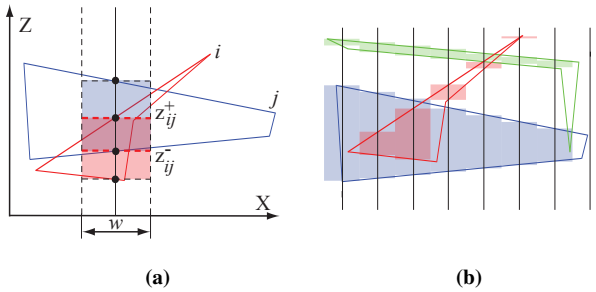


Figure 2: *Image-based volume contact.* (a) *Principle. Volumes and their intersections are discretized using depth intervals within a pixel.* (b) *Limitations. The sharp green feature intersecting the blue object is not captured by any ray, neither is the intersection between the smooth thin red and green parts. The intersection of the red and blue objects appears totally nested, thereby generating no repulsion force.*

2.2 Image-based volume contact

The volume contact approach proposed by [Faure et al. 2008; Allard et al. 2010] eases the computation of repulsion forces by minimizing volumes instead of distances. Volumes are approximated by discretizing the volume with rectangular boxes aligned to pixels in an image that is conceptually similar to a multi-layer depth buffer. The sampling can be done in any direction, but for simplicity we assume an orthogonal projection along z. Intersections are detected based on depth intervals, and their volume is defined as

$$\mathcal{V} = w \left(\sum_{(i,j)\in\mathcal{S}_z^+} \mathrm{z}_{ij}^+ \; - \sum_{(i,j)\in\mathcal{S}_z^-} \mathrm{z}_{ij}^- \right), \qquad (1)$$

where w is the area of a pixel, sets $\mathcal{S}_z^+$ and $\mathcal{S}_z^-$ respectively contain the pixels of the upper and lower surfaces of the intersection volume, the depth values z_{ij}^+ and z_{ij}^- are measured between an object surface and and an arbitrary reference plane orthogonal to the sampling direction, as shown in Fig. 2a.

Repulsion forces to reduce intersection volumes can be computed using the derivative of the volume with respect to the vertex coordinates. The derivatives with respect to the coordinates along the viewing axis are

$$\frac{\partial \mathcal{V}}{\partial \mathbf{p}_k^z} = w \left(\sum_{(i,j)\in\mathcal{S}_z^+} \frac{\partial \mathrm{z}_{ij}^+}{\partial \mathbf{p}_k^z} \; - \sum_{(i,j)\in\mathcal{S}_z^-} \frac{\partial \mathrm{z}_{ij}^-}{\partial \mathbf{p}_k^z} \right), \qquad (2)$$

where $\mathbf{p}_k^z$ is the z coordinate of vertex number k. Conveniently, the scalar $\partial \mathrm{z}_{ij}(\mathbf{p})/\partial \mathbf{p}_k^z$ at a given pixel corresponds to the barycentric coordinate of the intersection point with respect to vertex k. The computation is performed in three orthogonal directions to compute

the derivatives with respect to each coordinate. Details can be found in [Faure et al. 2008; Allard et al. 2010].

Interestingly, the method returns a geometric model of the intersection, and it is completely agnostic to the physical models and the numerical methods. The traditional Signorini contact equations can be easily adapted by replacing the relative displacement along the normal direction with the derivative of the intersection volume given in Eq. 2 [Allard et al. 2010]. The volume-based contact model can thus be straightforwardly combined with virtually all the available force computation methods, penalty- or constraint-based, applied to deformable objects as well as rigid bodies. The repulsion forces are parallel to the direction of maximum volume change, which corresponds to the average surface normal in the contact area. Guaranteeing the precision of the intersection volume is important for sensitive detection and precise tracking of contact points, as demonstrated in the Results section. The precision of the volume derivative, not guaranteed by our method, impacts the direction of the repulsion force, with limited consequences on the final outcome. We note that in many cases the normal direction used in distance-based methods is also not precisely defined, and the friction properties are also approximate; we defer the study of bounds on the volume derivative to future work. Since we also provide the intersection point and its normal, our method can be easily integrated with other simulation methods.

3 Method Overview

In this section we present an overview of blueour volume intersection computation in the simpler 2D setting, since it is easier to illustrate the core ideas. Generalization to the 3D setting is provided in subsequent sections. To make the transition between the two easier, we will refer to 2D edges as "faces" and 2D vertices as "edges" – this is equivalent to extruding the 2D polygons perpendicular to the sheet of the paper.

In previous work [Faure et al. 2008; Allard et al. 2010], intersection volumes were computed on a uniform grid. This can lead to under sampling in some areas (e.g., missed intersections at the right of Fig. 2b) and oversampling in other flat regions. The goal of this paper is to compute the intersection volume of polyhedra at arbitrary precision. We replace the rasterization with the casting of parallel rays. blueFor simplicity and without loss of generality, we assume that the ray direction is z, and defer the discussion on selecting good directions to Section 4.4. Each ray is associated with a cross-section, called a "tile", with area w. Thus a ray should be thought of as a box swept by the tile in the direction of the ray, which we will refer to as the *ray volume*. When the ray volume intersects only a single face, intersection volume is exactly computed by midpoint quadrature. Otherwise, there is an error in the volume estimate due to adjacent faces in the same ray volume, that we can bound as shown next.

3.1 Volume Error

In Fig. 3a a ray volume centered on R contains an edge $\mathbf{e}^{ij}$ generated by faces F^i and F^j, responsible for a volume error highlighted by the green upward-hatched triangle. This error is maximized when the ray intersects $\mathbf{e}^{ij}$ (ie, if R is at R' instead); this gives us a bound on the volume error, δ^{ij}. In the figure this is highlighted using red downward hatches. The same analysis holds for a silhouette edge, shown in Fig. 3b.

For a single edge the volume error depends on the size of the tile, w, as well as the slopes of two faces, $\alpha^i = n_x^j/n_z^j$ and $\alpha^j = n_x^i/n_z^i$, where $\mathbf{n}$ denotes the outward normal of a face. In this simple case

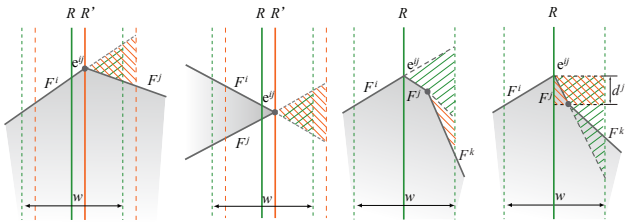


Figure 3: *blue 2D Volume approximation. In each figure, a solid vertical line represents a ray, and dotted lines of the same color mark the extent of the ray volume. F, $\mathbf{e}$ are face and edge's 2D representation. Each volume error created by an edge $\mathbf{e}^{ij}$ is highlighted with a hatched polygon with the same color as the ray.* (a)*: The volume is maximum (in red) when the ray crosses the edge.* (b)*: This also holds for silhouette edges.* (c)*: Multiple edge volumes (red, green) are summed up.* (d)*: Narrow face volumes are more tightly bound by rectangles (red) than by triangles (green).*

it is easy to see from the equation of a triangle's area that

$$\delta^{ij} = \frac{1}{2}(k^i|\alpha^i| + k^j|\alpha^j|)(\frac{w}{2})^2, \tag{3}$$

where k is the sign of face's contribution, determined by Alg. 1.

Algorithm 1 F^i, F^j's volume error sign. ϵ is a small real number.

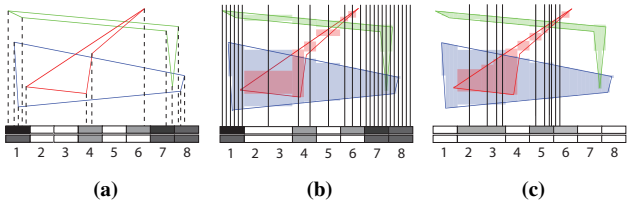
function $\mathscr{F}_k(\mathbf{n}^i, \mathbf{n}^j, \alpha^i, \alpha^j)$

if $|\mathbf{n}_z^i| > \epsilon$ & $|\mathbf{n}_z^j| > \epsilon$ **then** ▷ general case
  if $\alpha^i\alpha^j \geq 0$ **then**
    $k^i = 1; k^j = 1$
  else if $|\alpha^i| > |\alpha^j|$ **then**
    $k^i = 1; k^j = -1$
  else
    $k^i = -1; k^j = 1$
else if $|\mathbf{n}_z^j| < \epsilon$ **then** ▷ F^j is parallel with ray
  if $\mathbf{n}^i\mathbf{n}^j \geq 0$ **then**
    $k^i = -1; k^j = 1$
  else
    $k^i = 1; k^j = 1$
else if $|\mathbf{n}_z^i| < \epsilon$ **then** ▷ F^i is parallel with ray
  if $\mathbf{n}^i \cdot \mathbf{n}^j \geq 0$ **then**
    $k^i = 1; k^j = -1$
  else
    $k^i = 1; k^j = 1$

More generally, it is possible to have multiple edges in the same ray volume, as shown in Fig. 3c; a bound on the volume error is given by the sum of volume errors associated with each edge.

In Fig. 3d the edge is connected to a narrow face F^j. The extrapolation of the face to the half width of the pixel creates a much larger bound than the real error. Moreover, when F^j becomes nearly parallel to the ray, α^j in Eq. 3 would result in excessively large or infinite bounds. In these cases, the error is more tightly bounded by a box with its depth adjusted to the face depth interval d^j and the corresponding edge error bound is:

$$\delta^{ij} = \frac{w}{2}k^j d^j + \frac{1}{2}k^i|\alpha^i|(\frac{w}{2})^2. \tag{4}$$



(a) (b) (c)

Figure 4: *Overview of our method, in 2D.* (a) *Edge rasterization and EBPI computation.* (b) *Optimized sampling density based on the EBPI. The discretized volumes are denoted using rectangular areas.* (c) *Refinement for transient edges.*

3.2 Error Bound Polynomial Image

Each edge of the polygon contributes to the ray volume error bound as a function Δ of the tile extent. By combining Eq. 3 and 4

$$\Delta(w) = w^2 \sum_{(i,j)\in\mathcal{E}} \frac{1}{8}(k^i\alpha^i + k^j\alpha^j) + w \sum_{(i,j)\in\mathcal{E}} (k^i d^i + k^j d^j), \quad (5)$$

where $\mathcal{E}$ is the set of adjacent face index pairs whose common edges cross the ray volume; for each face either α or d equals to zero (Eq. 3 and 4). This function has two important features which generalize to the 3D case. First, it is a low degree polynomial in the dimensions of the tiles. Second, it depends only on the geometry of the edge neighborhood and the ray direction. Therefore we propose to represent it with a 2D image, which we called Error Bound Polynomial Image (EBPI), with the coefficients of Δ stored in separate color channels. The EBPI is efficiently constructed in parallel by edge rasterization.

For this 2D case, the EBPI is illustrated in Fig. 4 as the two-row grid on the bottom. The upper and lower rows contain the coefficients of w and w^2, respectively. The eight pixels for both channels contain various colors after rasterization, shown in 4a, depending on the geometry above them. Pixels $2, 3, 5$ are empty because they contain no edges. Pixels $1, 7, 8$ are very dark for the second channel since they contain steep geometry; the corresponding channel in pixel 6 is empty because it contains no narrow face.

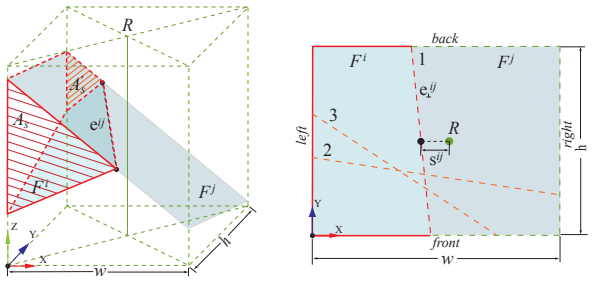
3.3 Adaptivity

Given a desired precision criterion, an optimal oversampling resolution can be achieved by solving Eq. 5 for w, as illustrated in Fig. 4b. The tiling is obtained by subdivision or gathering of the pixels. If more than one ray is needed, a regular subdivision is computed (pixels 1, 6, 7 and 8). Conversely, pixels can be recursively gathered if the geometry is simple enough (pixels 2, 3).

Transient edges created by mesh intersections are not explicitly represented in the meshes. However, the EBPI provides us with upper and lower bounds of the surfaces around the intersection points. This allows us to detect possible transient edges, to update the EPBI and shoot rays accordingly, as illustrated in Fig. 4c (pixels $2-3$, 5 and 6). More details are discussed in Section 5.

4 3D Volume Adaptive Sampling

In this section, we will extend the previous derivations to the 3D case and expand the method for obtaining the optimal subsampling resolution in 3D, with rays along the z direction again.



(a) (b)

Figure 5: *3D Volume approximation. A solid vertical line represents a ray, and dotted lines of the same color mark the extent of the ray in the two orthogonal directions.* (a) *The volume error is the frustum outlined with red color. Areas A_s and A'_s, highlighted with red hatched triangles, are on the two base planes of the frustum.* (b) *Top view of the pixel. $\mathbf{e}^{ij}_\perp$ is the projection of $\mathbf{e}^{ij}$ on the XY plane; s^{ij} is the distance between R and $\mathbf{e}^{ij}_\perp$ along the X direction.*

4.1 Edge error

The 3D case is illustrated in Fig. 5, where the tile size of R is (w, h), assuming that $\mathbf{e}^{ij}$ crosses the front and back boundaries of the ray volume, $\mathbf{e}^{ij}_\perp$ is the projection of $\mathbf{e}^{ij}$ on the XY plane. The edge volume error is the red frustum, surrounded by the pixel boundary surfaces and extended mesh surfaces. It can be calculated as $V = \frac{h}{3}\left(A_S + A'_S + \sqrt{A_S A'_S}\right)$, where A_S and A'_S are the base areas of the frustum. Consequently, we get

$$V = \frac{h}{6}(k^i|\alpha^i| + k^j|\alpha^j|)\left[3(\frac{w}{2} - s^{ij})^2 + (\frac{h}{2}\beta^{ij})^2\right], \quad (6)$$

where s^{ij} is the distance from the tile center to $\mathbf{e}^{ij}_\perp$ along the X direction, $\beta^{ij} = |n^j_y n^i_z - n^i_y n^j_z| / |n^j_x n^i_z - n^i_x n^j_z|$ is the slope of $\mathbf{e}^{ij}_\perp$ in the XY plane; α^i and α^j have the same definition as in the 2D case.

Since the maximum error is reached when the edge passes through the center of the tile($s^{ij} = 0$), the volume error bound δ^{ij} is

$$\delta^{ij} = \frac{h}{6}(k^i|\alpha^i| + k^j|\alpha^j|)\left[3(\frac{w}{2})^2 + (\frac{h}{2}\beta^{ij})^2\right]. \quad (7)$$

The projection of the edge direction to the XY plane can intersect the pixel boundary in three different ways labeled 1, 2, and 3 in Fig. 5b. Eq. 7 corresponds to case (1). A similar equation is obtained for case (2) when $\mathbf{e}^{ij}$ crosses the left and right boundaries, listed in Alg. 2. In the case labeled 3, the edge crosses two adjacent boundaries of the pixel, the maximum volume error also occurs when the edge crosses the ray. This corresponds to the rectangle diagonal, which can be handled by either case 1 or case 2. Finally, the box error bound for narrow face (Fig. 3d) is extended as a 3D box with depth adjusted to the maximum depth of the face within the pixel. We empirically find that comparing the actual depth interval of the face with $\frac{1}{4}$ of the frustum-extrapolated depth is a reasonable criterion, and defer a more precise study to future work.

Putting it all together, the general error bound associated with each face of an edge is given by Alg. 2. In function $\mathscr{F}_\delta$, the subscripts 1, 2 of α and β correspond to cases 1 and 2 of Fig. 5b, respectively.

Algorithm 2 Error bound created by F^i of $\mathbf{e}^{ij}$, within a pixel of size $w \times h$. $\mathbf{r}$ is the sampling ray direction.

function $\mathscr{F}_\delta(\mathbf{n}^i, \mathbf{n}^j, w, h)$

$\beta_1^i = \beta_2^i = \alpha_1^i = \alpha_2^i = 0$ ▷ Initialize
$d^i = F^i$ *depth interval*
compute k^i with Alg. 1

if $\mathbf{n}^i \cdot \mathbf{r} = 0$ **then** ▷ F^i is steep
  $\delta = \frac{1}{2}k^i whd^i$
else if $\mathbf{n}^i \cdot \mathbf{n}^j = 1$ **then** ▷ F^i, F^j are coplanar
  $\delta = 0$, $d^i = 0$
else if $\frac{l_y^i}{l_x^i} \geq \frac{h}{w}$ **then** ▷ F^i in case 1
  $\alpha_1^i = \frac{n_x^i}{n_z^i}$, $\beta_1^i = |\frac{n_y^j n_z^i - n_y^i n_z^j}{n_x^j n_z^i - n_x^i n_z^j}|$
  if $\frac{1}{8}w|\alpha_1^i| > d^i$ **then** ▷ F^i is narrow
    $\delta = \frac{1}{2}k^i whd^i$, $\beta_1^i = \alpha_1^i = 0$
  else
    $\delta = \frac{h}{6}k^i|\alpha_1^i|\left[3(\frac{w}{2})^2 + (\frac{h}{2}\beta_1^i)^2\right]$, $d^i = 0$
else ▷ F^i in case 2
  $\alpha_2^i = \frac{n_y^i}{n_z^i}$, $\beta_2^i = |\frac{n_x^j n_z^i - n_x^i n_z^j}{n_y^j n_z^i - n_y^i n_z^j}|$
  if $\frac{1}{8}h|\alpha_2^i| > d^i$ **then** ▷ F^i is narrow
    $\delta = \frac{1}{2}k^i whd^i$, $\beta_2^i = \alpha_2^i = 0$
  else
    $\delta = \frac{w}{6}k^i|\alpha_2^i|\left[3(\frac{h}{2})^2 + (\frac{w}{2}\beta_2^i)^2\right]$, $d^i = 0$

4.2 Error bound polynomial image

Let us assume that each pixel of the rasterized edge image has size $w_b \times h_b$. If we oversample it using tiles of size w, h, the total error within the rasterized pixel is the sum of the contributions of all the edges across all the tiles:

$$\Delta(w,h) = \sum_{[i,j]\in\mathcal{E}} (\frac{l_x^{ij}}{w} + \frac{l_y^{ij}}{h})\mathscr{F}_\delta(\mathbf{n}^i, \mathbf{n}^j, w, h), \tag{8}$$

where l_x and l_y are the projection lengths of the edge along the X and Y directions within the pixel; $\mathcal{E}$ is the set of edges in the pixel. The coefficient before $\mathscr{F}_\delta$ represents the maximum number of sub-pixels crossed by $\mathbf{e}^{ij}$, while $\mathscr{F}_\delta$ is the error function defined in Alg. 2, Both the length and projected slope difference of the edge are taken into consideration for the volume error bound, so mesh tessellation does not lead to over-conservative bounds. The expansion of Eq. 8 provides us with the following error bound function:

$$\begin{aligned}\Delta(w,h) =& w^2 \sum[\frac{1}{8}k|\alpha_1|l_y + \frac{1}{24}k|\alpha_2|{\beta_2}^2 l_x] \\ &+ h^2 \sum[\frac{1}{8}k|\alpha_2|l_x + \frac{1}{24}k|\alpha_1|{\beta_1}^2 l_y] \\ &+ wh \sum[\frac{1}{8}k|\alpha_1|l_x + \frac{1}{8}k|\alpha_2|l_y] \\ &+ \frac{h^3}{w}\sum(\frac{1}{24}k|\alpha_1|{\beta_1}^2 l_x) \\ &+ \frac{w^3}{h}\sum(\frac{1}{24}k|\alpha_2|{\beta_2}^2 l_y) \\ &+ h\sum\frac{1}{2}dl_x + w\sum\frac{1}{2}dl_y.\end{aligned} \tag{9}$$

where the edge indices are ignored for clarity. This polynomial has seven terms. The first five correspond to cases (1) and (2) in Fig. 5b, while the last two correspond to narrow faces. Therefore, seven color channels are needed to store the coefficients in the EBPI.

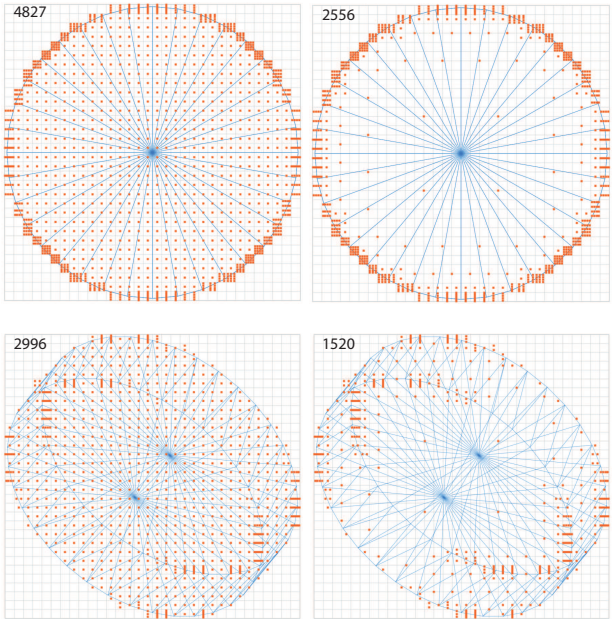


Figure 6: *Ray distribution for sampling a cylinder. The red points denote rays which are perpendicular with the paper. The total number of rays along three orthogonal directions is given on top left of each figure.* Left: *uniform grid with local refinement.* Right: *multi resolution coarsening is used.* Top: *worst sampling directions.* Bottom: *optimized directions.*

4.3 Pixel refinement and coarsening

Unlike the 2D case, the oversampling resolution can not be directly computed from Eq. 9, since it is underdetermined. To meet the precision criterion while minimizing the number of rays, we formulate the problem as a constrained optimization:

$$\begin{aligned}\underset{w,h}{\text{maximize}} \quad & wh \\ \text{subject to} \quad & 0 \leq w \leq w_b,\ 0 \leq h \leq h_b, \\ & \Delta(w,h) \leq \bar{\Delta}\end{aligned} \tag{10}$$

where the objective function maximizes each sub-pixel's area, which is equivalent to minimize the number of rays; w_b and h_b are the EBPI tile size, and $\bar{\Delta}$ is the desired volume tolerance per pixel.

The convexity of the objective function makes this nonlinearly constrained optimization relatively easy to solve. The first two constraints reduce the search space to one dimension. When the pixel contains only one edge and the third constraint is active, the optimal aspect ratio of the sub-pixel equals to the slope of the edge in the image plane, $\gamma = \frac{w}{h} = \frac{l_x}{l_y}$. This is easily derived by substituting w with γh, and solving the equation for γ using a Lagrange multiplier. Intuitively speaking, the edge occupies fewer tiles if it is parallel to their diagonal, and the estimated volume error bound is closer to the real volume error. When the EBPI pixel contains multiple edges, each edge makes a different contribution to the volume error, and the optimal aspect ratio is given by a weighted sum ratio:

$$\gamma = \frac{w}{h} = \frac{\sum l_x(|\alpha_1| + |\alpha_2| + d)}{\sum l_y(|\alpha_1| + |\alpha_2| + d)}. \tag{11}$$

This allows us to reduce the convex optimization to one-dimensional root findings, performed using the Newton method.

Once the optimal pair (w, h) is computed, the subdivision factors as the smallest integers greater than or equal to w_b/w and h_b/h. A basic example of pixel refinement using the EBPI is illustrated in the top left of Fig. 6. Refinement occurs near the silhouette, with different resolutions in horizontal and vertical directions, depending on the local orientation of the silhouette.

Since the subdivision resolution is uniform across a given EBPI pixel, a single sharp detail within a large pixel may generate numerous rays. It is thus desirable to use a relatively fine EBPI resolution. However, to avoid large numbers of rays in relatively smooth regions, a locally coarse EBPI is desirable. To exploit the EBPI at multiple resolutions, we compute a multi resolution pyramid. Each pixel color in a coarse level gets the sum of the four corresponding pixel colors at the finer level, then the optimization Eq. 10 is performed. This adaptive approach allows us to use fine reference resolutions while sampling smooth regions sparsely. An example of pixel coarsening is shown in Fig. 6 using the same EBPI resolution (32×32) and desired volume error bound. EBPI coarsening effectively reduces the number of rays, especially in the smooth regions.

4.4 Viewing directions

Faces parallel with the sampling rays generate large errors, which in turn require large numbers of rays to guarantee precision. However, the three orthogonal directions used to sample the volume and its derivatives can be rotated arbitrarily. To avoid sampling directions parallel to large faces, we compute a histogram of the normal vectors, scaled by face area. This discretized version of the Extended Gaussian Image (EGI) [Horn 1984] of the geometry has been previously applied to the registration of 3D models using tessellated spheres, by looping over the polygons and testing their normal against tetrahedra [Dold 2005]. We leverage the accelerated environment mapping methods available in modern graphics APIs to efficiently compute the histogram as a cube map on the GPU. Each triangle accumulates its area to the texture pixel corresponding to its normal. Our orientation error function penalizes the orthogonality to the faces, since orthogonality with one axis implies parallelism with the two others:

$$e(\mathbf{q}) = \sum_{i \in \mathcal{T}} \sum_{j \in 0..2} \left(\widehat{\mathbf{n}}_i . \mathbf{a}_j(\mathbf{q})\right)^4, \qquad (12)$$

where the $\widehat{\mathbf{n}}_i$ are the weighted normals in texture $\mathcal{T}$ and the $\mathbf{a}_j$ are the three orthogonal directions transformed from the original world frame by quaternion $\mathbf{q}$. The study of an optimal cost function is deferred to future work.

We perform the optimization using the Fletcher-Reeves method [Press et al. 2007], combined with small random steps to exit local extrema. We leverage time coherency to efficiently update the rotation at each time step. Examples are shown in the bottom of Fig. 6. All the sampling directions are away from the surface normals, and the same volume precision is obtained using a smaller number of rays.

5 Polyhedron intersections

When two polyhedra intersect each other, transient edges are created by pairs of originally non-adjacent surfaces. Their corresponding volume error cannot be accounted for through edge rasterization. This may result in violations of the precision criterion, and some intersections may even be left undetected, as shown in the left of Fig. 7a. Computing explicitly the intersection edges based on the

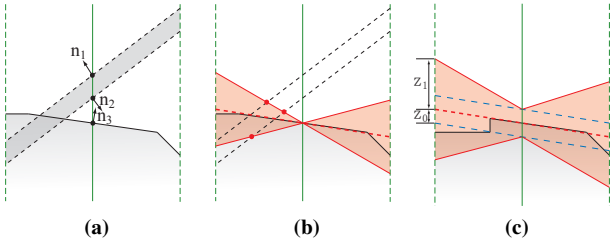


Figure 7: *Ray refinement for transient edges.* (a) *A thin object intersection near the boundary of a pixel, which could be totally missed based on ray intersection points.* (b) *Uncertainty region for the third intersection point and approximation planes for the other two points. Intersections between the uncertainty region and the approximation planes indicate the existence of transient edges.* (c) *Surface uncertainty region with a narrow face.*

faces would considerably decrease the performance of the method, while the advantage of image-based methods is precisely to avoid such geometric computations. Fortunately, the EBPI allows us to reliably infer the presence of transient edges using the volume error bound, and to apply a refinement strategy to maintain the precision.

blue When a ray hits a surface, the intersection point and its normal define a local approximation plane of the surface in the ray volume, illustrated using a red dashed line in Fig. 7b. If there are only coplanar edges in the ray volume, the approximation plane is identical to the actual plane. Otherwise, due to the slope difference of adjacent mesh faces, the difference between the actual surface and the approximation plane is within a butterfly-shaped region, highlighted in red in Fig. 7b. The distance is maximum at the ray boundary, $z_1 = \frac{1}{2}\sqrt{w^2 + h^2} \sum(|\alpha_1| + |\alpha_2|)$. If the ray volume contains narrow faces (i.e., $\sum d > 0$), there should be a constant offset $z_0 = \sum d$ for the whole region, as illustrated in Fig. 7c.

If another approximation plane (black dashed line in Fig. 7b) generated by other intersection point intersects with the butterfly region within the ray volume, we can conservatively deduce there is a transient edge (red dot in Fig. 7b). Its corresponding volume error can be estimated and accumulated based on the normal vectors of the intersection point and region boundary surface, like for a mesh edge. After traversing all the intersection points, if the updated volume error is larger than the precision threshold, a new subdivision resolution is computed, and new rays are scheduled. Otherwise, the list of ray-surface intersections is used to detect and model the volume intersections in the pixel. We can directly compute the resolution of a uniform subdivision to achieve the desired accuracy. However, since the ray may be at the center of a large tile, a uniform subdivision may result in a large number of additional rays. We thus oversample using an arbitrary resolution of 4×4 and perform the transient edge detection again, which may require several additional passes. Since each additional pass on the GPU has some latency, there is a trade-off to be made; we defer this study to future work.

One example is shown in Fig. 8, where a diamond-shaped polyhedron intersects a cube, and the geometry edges are depicted as blue and green lines. Our ray refinement scheme detects the existence of transient edges and appropriately increases the sampling density.

6 Implementation

The overall control flow of our method is summarized in Alg. 3. We use the CUDA [NVIDIA 2007] GPU library to compute the geometry terms of Alg. 2 in each direction, and store them in OpenGL

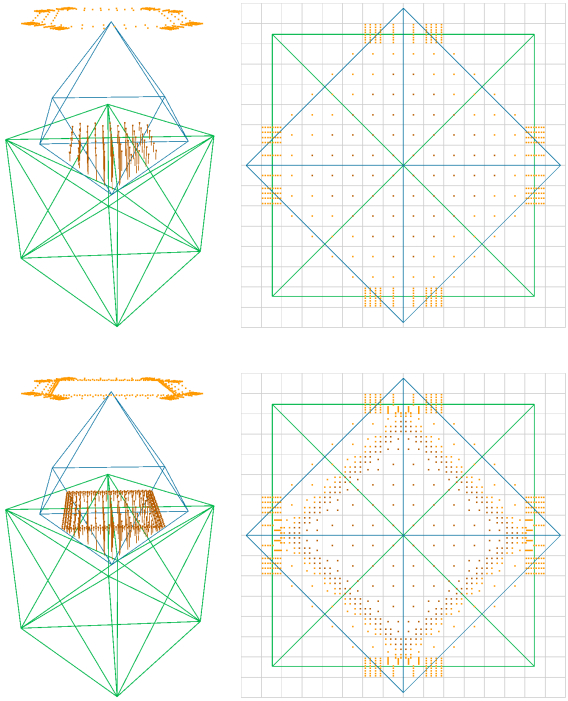


Figure 8: *Ray distribution with surface intersection, in one direction. Intersections are depicted as vertical lines, while intersection-free rays appear as points on the top horizontal plane.* Top: *without surface intersection refinement.* Bottom: *with surface intersection refinement.* Left: *perspective view.* Right: *ray direction view.*

vertex buffers. Then, the rasterization edges to the EBPI is performed by an OpenGL fragment shader, including the interpolation of d and the computation and accumulation of the terms of Eq. 9. Finally, the multi resolution pyramid and the sampling resolution in each pixel of the EBPI are computed using CUDA.

Parallel ray casting is performed using the GPU-based OptiX ray tracing engine [Parker et al. 2010]. Each ray's original point, direction and corresponding area are needed to correctly perform adaptive intersection volume sampling. We deployed OpenGL pixel buffer objects, which can be accessed by OpenGL, CUDA and OptiX, to share this necessary ray information. This allows Alg. 3 to be performed on GPU except the optimization of the viewing directions. Only the total number of rays is read back from GPU to CPU for OptiX. In case of surface intersections (Section 5), additional rays can be fired directly by the callback programs, but we found that deferring them to an additional schedule list is more efficient.

Each ray is associated with a buffer to store the list of surface intersections, which includes distance to ray origin, object index, triangle index and barycentric coordinates. There is virtually no limit on the number of intersections along a single ray; even if the buffer is full due to the depth complexity, an exception could be raised to flush it. At the end of the detection, the intersection volumes and their gradients are transmitted to the CPU to simulate contact response. We use a constraint-based method when very small inter-penetration depth or Coulomb friction are needed, or an implicit penalty-based method otherwise. After the animation step, we can directly re-use the ray-casting acceleration data structures in an OptiX-based ray-tracing renderer.

Algorithm 3 Control flow of our method.

```
Optimize viewing directions                                  ▷ Sec. 4.4
for all 3 orthogonal directions do
    for all mesh edges do                                    ▷ Sec. 4.1
        Compute α, β and d in Alg. 2
    Rasterize edges into EBPI                                ▷ Sec. 4.2
    Compute ray density for each EBPI pixel                  ▷ Sec. 4.3
    Coarsen pixels                                           ▷ Sec. 4.3
while ray list not empty do
    for all rays do
        Shoot the ray
        for all ray-surface intersections do
            Store intersection point in ray's payload
        Detect transient edges                               ▷ Sec. 5
        if need refinement then
            Create new rays
Do object intersection test
Transfer volumes and volume gradients to the CPU
```

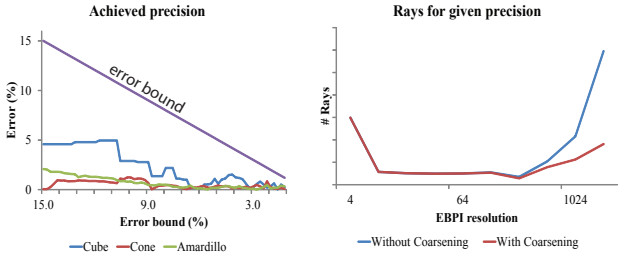


Figure 9: *Precision and ray number tests.* Left: *Achieved vs. desired precision of volume for three different polyhedra.* Right: *The number of rays used to obtain a given precision, depending on the EBPI resolution, with coarsening (red) or without (blue).*

7 Results

7.1 Adaptivity and precision

The experiments illustrated in the left of Fig. 9 confirm that the EBPI allows us to achieve the desired precision. Three polyhedra were tested. A cube, with faces aligned with the ray directions, generates the highest error due to its deep steep faces. A sharp cone creates less error, even though its thin tip is challenging. Our method is even more conservative for a wide variety of practical models, including the armadillo tested in this experiment.

In the right of Fig. 9, we compare the number of rays necessary to achieve a desired precision, depending on the EBPI resolution. Low resolutions are not efficient, since their large pixels are oversampled uniformly, even if only a local detail requires a high resolution. High resolutions can be inefficient too, due to the large number of pixels, but our coarsening strategy mitigates this problem. The optimal number of rays is obtained for a wide range of moderate resolutions, which makes our method easy to tune and keeps its memory footprint low.

The transient edges, not explicit in the geometry, are bracketed and refined by our algorithm. In the accuracy test shown in Fig. 10, the precision criterion is met only if we turn on ray subdivision. In the first example, the object interpenetration occurs between large flat regions, so there is no subdivision of the pixels around that area, while for the concave example, the edges of the cross shaped object introduce more rays, which result in the total precision increase. In both cases, adaptivity is necessary to guarantee the precision.

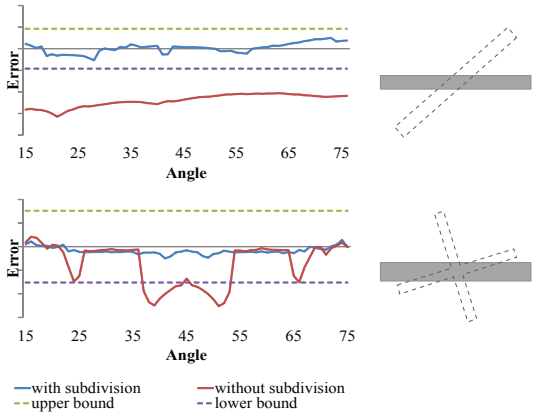


Figure 10: *Precision comparison of intersection volume for convex and concave objects. The objects outlined with dashed lines are rotated against a fixed object, represented by a solid box. The dashed lines in the left diagrams represent the upper and lower expected error bounds, set to 10% of the block volume. The achieved precision shows that ray refinement detects the transient edges and maintains the precision criterion.*

7.2 Performance

blue Our method can be roughly divided into three stages. The first one includes rastering edges into EBPI and computing non-uniform sampling density for each EBPI pixel. Its time complexity is $O(p + r^2)$, where p is the number of polygons and r is the EBPI resolution. Ray-surface intersection tests, along with updating the OptiX bounding volume hierarchies, are implemented in the second stage; this performance is determined by OptiX. The final stage is responsible for sorting ray-surface intersection points and calculating intersection volume and gradient; this has time complexity $O(n \log n)$, where n is the number of rays. The memory footprint includes the bounding volume hierarchy of the ray-tracer, which is $O(p \log p)$; the EBPI texture image, which is $O(r^2)$; and the volume gradient, whose size is proportional to the number of vertices on the surfaces of the intersection volumes.

We have experimentally checked by tessellating analytical shapes at various resolutions that the number of rays depends more on intrinsic shape properties than on surface resolution. This is an advantage of our approach compared with traditional methods which rely on basic primitive tests. Computation times are shown in Table 1. Though there is plenty of room for optimization in our prototype implementation, our computation times are comparable with [Tang et al. 2011], who report about $50ms$ for $100k$ triangles. The precise contact in the nut and bolt simulation requires more rays than the armadillo simulation, while the latter involves a larger number of triangles, thus more rasterization time.

We have tested our method on a set of challenging scenarios shown in the first page. Fig. 1a: The nut and bolt include a total of 332k triangles. Our contact model resolves the initial penetration due to a wrong relative orientation. To demonstrate the precision of our contact model, we apply a frictionless repulsion to allow the nut to spontaneously unscrew under the action of gravity. Fig. 1b: The very small obstacle hit by the cube could not be detected at uniform resolution, due to the comparatively large area of the contact between the cube and the ground, whereas our method accurately models it. However, since this is not a continuous collision detection method, the cube would skip the obstacle in case of large velocity. Fig. 1c: The example of the Ruina Wheels [Ruina et al. 2005] shows that objects with subtle, nearly invisible geometric differences can have very different behaviors, and that our precise contact model enables us to capture this difference. These two quasi-polygons have slightly convex and slightly concave faces, respectively. Each edge of the concave wheel hits the ground and dissipates energy, while the contact point progressively moves from one edge to the other on the convex wheel, allowing him to roll on a longer distance, as observed in the real world and shown in the accompanying video. Our adaptive method successfully simulates these different behaviors, as shown in the simulation, even though we are using a simple impulse-based method with post-stabilization. Fig. 1d: The colliding armadillos include a total of nearly 5 million triangles. Though implementation issues currently limit the size of our simulations, we believe that we will soon be able to handle much more complex models, thanks to the low complexity of our method. Fig. 1e: An interactive simulation featuring ray-traced images created using the same acceleration structures as in collision detection. It runs at 5 fps with ray-tracing and 40 fps using OpenGL rendering.

scene(tri #)	raster	BVH	casting(ray #)	read back
Bolt(332k)	54.3	8.9	20.0(92k)	52.8
Armas(1.2M)	112.4	25.1	6.9(44k)	17.5
Armas(1.6M)	165.8	29.5	7.1(43k)	20.0
Armas(1.9M)	192.8	36.6	7.7(43k)	19.7
Armas(3.4M)	319.8	69.2	7.0(61k)	37.6
Armas(4.4M)	415.9	92.3	7.8(42k)	32.1

Table 1: *Computation times in ms for bolt and nut (first line), and colliding armadillos with 512×512 EBPI resolution. Column* raster *corresponds to the rasterization and the computation of the sampling resolution. Column* BVH *corresponds to the OptiX BVH updates. Column* casting *shows the number of rays in braces and the corresponding computation time. The last column displays the time for the read back of the volume gradient to the CPU.*

8 Conclusion

The accuracy of intersection volume is important for plausible collision response. In this paper we have presented the first image-based collision detection method that provides the controllability of intersection volume without explicit geometrical computation, and demonstrated its relevance for precise contact modeling. Its computation combines rasterization at moderate resolution with adaptive ray casting, which allows more precise contact modeling where needed and a reduced memory footprint. The ability to share the acceleration data structures with ray-tracers makes the method a natural choice for interactive ray-traced simulations, which will be increasingly popular thanks to the ever-growing power of GPU-based ray-tracers.

Although our method does not allow zero volume error, it can be maintained at an arbitrary low level. We stress that details are never missed during EBPI construction, but can be neglected if the value of the volume error threshold is too large. Similarly with previous image-based collision detection methods, watertight meshes are required.

Tuning the value of the threshold raises the question of assessing the quality of a physical animation, which should be addressed in future work. A straightforward extension of this framework would be to perform continuous collision detection using rays in the velocity directions. We will investigate less conservative error bounds. blueBounding contact force errors would also require bounds on the volume gradient. Moreover, our adaptive method mitigates but does not completely eliminate the general limitation of volume-based contact, namely the possibility that sharp features penetrate

objects without creating large intersection volumes. In most cases, sharp features are a small subset of object geometry and could be tagged as such, and we believe that our adaptive approach opens an avenue for the exploitation of specific information associated with geometric features.

Acknowledgements We would like to thank Hui Chang, François Jourdes and the anonymous reviewers for their help. This work was supported in part by the China Scholarship Council, the Canada Research Chairs Program, NSERC, the Human Frontier Science Program, and the Peter Wall Institute for Advanced Studies.

References

ALLARD, J., FAURE, F., COURTECUISSE, H., FALIPOU, F., DURIEZ, C., AND KRY, P. G. 2010. Volume contact constraints at arbitrary resolution. *ACM Trans. Graph. 29* (July), 82:1–82:10.

BACIU, G., AND WONG, W. S.-K. 2004. Image-based collision detection for deformable cloth models. *IEEE Transactions on Visualization and Computer Graphics 10*, 649–663.

BARBIČ, J., AND JAMES, D. 2007. Time-critical distributed contact for 6-dof haptic rendering of adaptively sampled reduced deformable models. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 171–180.

BARBIČ, J., AND JAMES, D. L. 2010. Subspace self-collision culling. *ACM Trans. Graph. 29* (July), 81:1–81:9.

DINGLIANA, J., AND O'SULLIVAN, C. 2000. Graceful degradation of collision handling in physically based animation. *Computer Graphics Forum 19*, 239–247.

DOLD, C. 2005. Extended gaussian images for the registration of terrestrial scan data. In *ISPRS WG III/3, III/4, V/3 Workshop "Laser scanning 2005"*.

FAURE, F., BARBIER, S., ALLARD, J., AND FALIPOU, F. 2008. Image-based collision detection and response between arbitrary volume objects. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 155–162.

GALOPPO, N., OTADUY, M. A., MECKLENBURG, P., GROSS, M., AND LIN, M. C. 2006. Fast simulation of deformable models in contact using dynamic deformation textures. In *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 73–82.

GOVINDARAJU, N. K., KNOTT, D., JAIN, N., KABUL, I., TAMSTORF, R., GAYLE, R., LIN, M. C., AND MANOCHA, D. 2005. Interactive collision detection between deformable models using chromatic decomposition. *ACM Trans. Graph. 24* (July), 991–999.

HABLE, J., AND ROSSIGNAC, J. 2005. Blister: Gpu-based rendering of boolean combinations of free-form triangulated shapes. *ACM Trans. Graph. 24* (July), 1024–1031.

HARMON, D., PANOZZO, D., SORKINE, O., AND ZORIN, D. 2011. Interference-aware geometric modeling. *ACM Trans. Graph. 30* (Dec.), 137:1–137:10.

HEIDELBERGER, B., TESCHNER, M., KEISER, R., MULLER, M., AND GROSS, M. 2004. Consistent penetration depth estimation for deformable collision response. In *Proc. VMV*, 339–346.

HEIDELBERGER, B., TESCHNER, M., AND GROSS, M. 2004. Detection of collisions and self-collisions using image-space techniques. In *Proc. WSCG*, 145–152.

HERMANN, E., FAURE, F., AND RAFFIN, B. 2008. Ray-traced collision detection for deformable bodies. In *3rd International Conference on Computer Graphics Theory and Applications*, 293–299.

HORN, B. K. P. 1984. Extended gaussian images. *Proc. of the IEEE 72*, 1671–1686.

JE, C., TANG, M., LEE, Y., LEE, M., AND KIM, Y. J. 2012. Polydepth: Real-time penetration depth computation using iterative contact-space projection. *ACM Trans. Graph. 31*, 5:1–5:14.

MENON, J., MARISA, R. J., AND ZAGAJAC, J. 1994. More powerful solid modeling through ray representations. *IEEE Comput. Graph. Appl. 14* (May), 22–35.

NVIDIA. 2007. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation.

PARKER, S. G., BIGLER, J., DIETRICH, A., FRIEDRICH, H., HOBEROCK, J., LUEBKE, D., MCALLISTER, D., MCGUIRE, M., MORLEY, K., ROBISON, A., AND STICH, M. 2010. Optix: a general purpose ray tracing engine. *ACM Trans. Graph. 29* (July), 66:1–66:13.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 2007. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*, 3 ed. Cambridge University Press.

RUINA, A., BERTRAM, J. E., AND SRINIVASAN, M. 2005. A collisional model of the energetic cost of support work qualitatively explains leg sequencing in walking and galloping, pseudo-elastic leg behavior in running and the walk-to-run transition. *Journal of Theoretical Biology 237* (Nov), 170–192.

SCHVARTZMAN, S. C., PÉREZ, . G., AND OTADUY, M. A. 2010. Star-contours for efficient hierarchical self-collision detection. *ACM Trans. Graph. 29* (July), 80:1–80:8.

SUD, A., GOVINDARAJU, N., GAYLE, R., KABUL, I., AND MANOCHA, D. 2006. Fast proximity computation among deformable models using discrete voronoi diagrams. *ACM Trans. Graph. 25* (July), 1144–1153.

TANG, M., MANOCHA, D., AND TONG, R. 2009. Multicore collision detection between deformable models. In *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, 355–360.

TANG, M., MANOCHA, D., LIN, J., AND TONG, R. 2011. Collision-streams: fast gpu-based collision detection for deformable models. In *Symposium on Interactive 3D Graphics and Games*, 63–70.

TESCHNER, M., KIMMERLE, S., HEIDELBERGE, B., ZACHMANN, G., RAGHUPATHI, L., FUHRMANN, A., CANI, M.-P., FAURE, F., MAGNENAT-THALMANN, N., STRASSER, W., AND VOLINO, P. 2004. Collision detection for deformable objects. In *Eurographics State-of-the-Art Report*, 119–139.