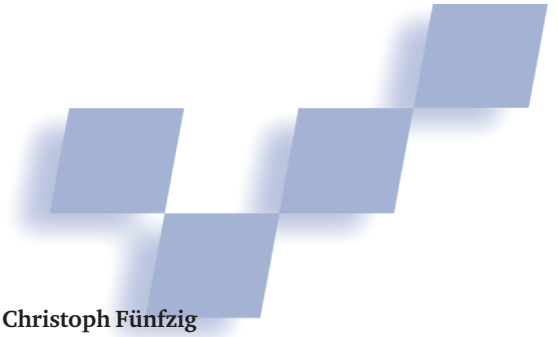
Hierarchical Spherical Distance Fields for Collision Detection



Christoph Fünfzig
Braunschweig University of Technology, Germany

Torsten Ullrich and Dieter W. Fellner
Graz University of Technology, Austria

This article presents a fast collision detection technique for all types of rigid bodies, demonstrated using polygon soups. The new approach uses spherical distance fields, which are stored in a compact representation.

The problem of collision detection between objects is fundamental in many different communities including CAD, robotics, computer graphics, and computational geometry.

We can classify the general problem of collision detection into three categories:

- collision detection, which tests whether two or more objects collide;
- collision determination, which determines which parts of some objects intersect; and
- collision response, which answers the question, Which action should be taken in response to a collision?

For practical collision detection, all approaches first consider the bounding volumes of all participating models. This broad phase only extracts potential collision pairs.[1] The following narrow phase performs a precise collision detection for each potential collision pair with the model representation. A general approach for all model representations uses hierarchies of simple bounding volumes containing model parts again. For more on other approaches, see the "Related Work" sidebar.

In this article, we present two algorithms for computing a discrete spherical distance field of models. For compactly storing the distance field, we use a subsampling filter bank.

Approach

In contrast to multiresolution representations with wavelets or progressive meshes, our approach is simpler and does not reproduce the model topology exactly, but generates conservative bounding volumes for model parts efficiently.

The spherical sampling according to a single center point requires a spherical parameterization, where we have chosen one with six charts derived from the box sides.[2] A spherical representation allows fast model rotation and the on-the-fly generation of spherical shell bounding volumes for model parts. Others have already used spherical shells as bounding volumes for spline models.[3,4] Krishnan et al. use this bounding volume in an oriented bounding box hierarchy to bound spline model parts more tightly.[4] The robotics literature has called the same bounding volume bi-sphere. Hamlin, Kelley, and Tornero present an algorithm for distance computation between bi-spheres and more general convex hulls of a finite number of spheres (S-topes), which resembles the Gilbert-Johnson-Keerthi algorithm for convex polytopes.[5] The new approach can be used in this area for path planning applications.

The spherical shell bounding volumes at each level of the hierarchical spherical distance field discard the parts in certain sphere sectors from further consideration. Depending on the application requirements, the algorithm can report at the leaf level a single triangle per spherical shell, a triangle per model layer inside a spherical shell, and all triangles inside a spherical shell. To the best of our knowledge, a spherical distance field has never previously been applied to collision detection despite its advantages for efficient rotation. Here, we show benchmarks of all variants of the collision determination and rank it against other techniques.

Spherical model representation

Like most collision-detection algorithms, our approach consists of two parts: a preprocessing step and a testing routine that represents the intrinsic collision test. Before we describe the algorithm, we present the main idea through a simple, 2D example.

Overview

During the preprocessing step, the algorithm takes an initial model and determines a model center. According to this center point, the model is transformed into spherical coordinates and sampled into several intervals. As a result, the algorithm stores the maximum radius and the minimum radius for each sample (see Figure 1a on p. 66).

The basic concept used in the preprocessing step works analogously to a wavelet transform. It starts with

 Published by the IEEE Computer Society

Related Work

A general approach for collision detection uses hierarchies of simple bounding volumes containing model parts. Researchers have proposed several bounding volumes including spheres, axis-aligned bounding boxes, oriented bounding boxes, and discrete orientation polytopes.[1] Here the performance depends on the tightness of the bounding volume, the efficiency of the intersection test for the bounding volume, and the strategy for hierarchy generation. Gottschalk, Lin, and Manocha give a complete description of the tree construction and collision test using oriented bounding boxes.[2] Recently, researchers have modified the approaches using bounding volume hierarchies for time-critical collision handling.[3]

Bounding volume hierarchies with deformable models require additional time for refitting,[3] and optimizations exist for special deformations.[4]

In the area of deformable models, researchers commonly use simpler structures like Cartesian grids and 1D arrays accessed by hashing. McNeely et al.[5] built up a voxel grid for the static model, where the points of a small movable model are queried against. This approach guarantees the high feedback rate needed by a haptic feedback device, and it is tailored to the haptic application domain. Fuhrmann et al.[6] also use a grid to store the Cartesian distance field for a static model draped with a deformable cloth model. Here, as with McNeely's approach, one model is completely static. Problems of these approaches are the large memory consumption and the necessity to choose the grid resolution beforehand. Accessing rotated Cartesian grids is a costly operation if done randomly and without exploiting coherence. Cartesian grids classically serve as a spatial data structure to structure the simulation domain. Teschner et al.[7] use hashing as a grid compression technique and perform collision detection between points and tetrahedral elements directly on the hashed array. The performance strongly depends on a uniform distribution of tetrahedral elements and points within the hashed array.

Other researchers consider image-space techniques for collision detection using graphics hardware. See Heidelberger, Teschner, and Gross[8] for a good summary. One approach is to perform ray casting through the volume of interest. It can be implemented on rasterizing graphics hardware using orthographic projection with the depth and stencil buffers. The resulting layered depth representation can be made robust against vanishing depth intervals, but requires slow buffer read-backs in multiple passes.[8]

Alternative approaches avoid buffer read-backs. Knott and Pai[9] detect edge points of one object inside another object by counting ray-surface intersections. The approach is quite fast, but it is not robust in case of occluded edges. Govindaraju et al.[10] use hardware-accelerated occlusion queries instead. The visibility tests allow for sorting out objects or object primitives, which do not participate in any collision within the set. Although the setup is complex it can report detailed collision information. The approaches using graphics hardware previously were fast at collision detection or point-in-volume tests, but had difficulties with collision determination, like extracting all colliding object parts.

References

1. L. Lin and D. Manocha, "Collision and Proximity Queries," *Handbook of Discrete and Computational Geometry*, 2nd ed., J.E. Goodman and J. O'Rourke, eds., CRC Press, 2004.
2. S. Gottschalk, M.C. Lin, and D. Manocha, "OBB-Tree: A Hierarchical Structure for Rapid Interference Detection," *Proc. ACM Siggraph*, ACM Press, 1996, pp. 171-180.
3. M. Teschner et al., "Collision Detection for Deformable Objects," *State of the Art Report*, Eurographics, 2004.
4. L. Kavan and J. Zara, "Fast Collision Detection for Skeletally Deformable Models," *Computer Graphics Forum*, vol. 24, no. 3, 2005, pp. 363-372.
5. W.A. McNeely, K.D. Puterbaugh, and J.J. Troy, "Six Degrees-of-Freedom Haptic Rendering Using Voxel Sampling," *Proc. Siggraph*, ACM Press, 1999, pp. 401-408.
6. A. Fuhrmann, G. Sobottka, and C. Groß, "Distance Fields for Rapid Collision Detection in Physically Based Modeling," *Proc. GraphiCon*, Keldysh Inst. of Applied Mathematics, 2003, pp. 58-65.
7. M. Teschner et al., "Optimized Spatial Hashing for Collision Detection of Deformable Objects," *Proc. Vision, Modeling, and Visualization*, Eurographics Assoc./Blackwell Publishing, 2003, pp. 47-54.
8. B. Heidelberger, M. Teschner, and M. Gross, "Detection of Collisions and Self-Collisions Using Image-Space Techniques," *Proc. Winter School of Computer Graphics*, 2004, pp. 145-152.
9. D. Knott and D.K. Pai, "CInDeR: Collision and Interference Detection in Real-Time Using Graphics Hardware," *Graphics Interface*, A K Peters, 2003, pp. 73-80.
10. N. Govindaraju et al., "CULLIDE: Interactive Collision Detection between Complex Models in Large Environments Using Graphics Hardware," *Proc. ACM Siggraph/Eurographics Workshop Graphics Hardware*, Eurographics Assoc./Blackwell Publishing, 2003, pp. 25-32.

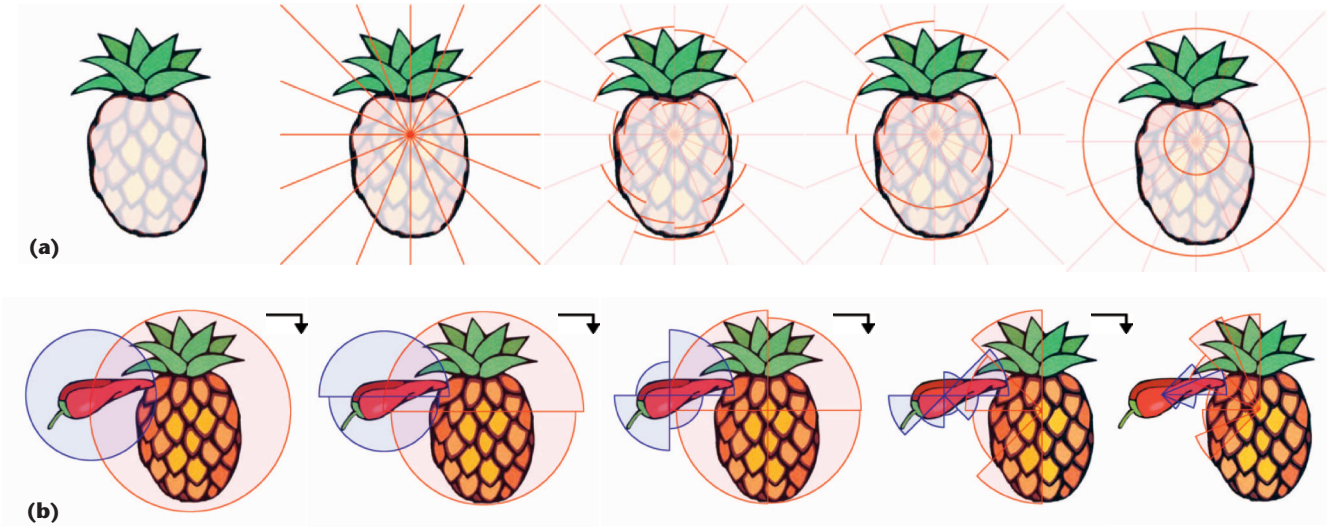
the high-resolution object and derives objects of lower resolution by storing the differences in detail coefficients to reverse this operation.

For correctness, a low-resolution model must bound all higher resolution models. Therefore, the Haar basis functions—which are used in image processing and describe an averaging and differencing process—are unsuitable.

For our purpose, maximizing and minimizing functions have to be taken as Figure 1a shows. Therefore, each object will be transformed into an inscribed circle and a circumference in the lowest resolution.

Having transformed all objects this way, we can perform a simple and fast collision test, which we will demonstrate with a pineapple and a red pepper (see Figure 1b). For clarity, we use solid objects with zero minimum radius in the illustration.

At runtime, the intersection test starts with the model representation at its lowest resolution and tests whether both models collide at this resolution. If this test is positive, the algorithm will increase the level of detail. Therefore, it's important for performance that only intersecting sectors are considered further on. Figure



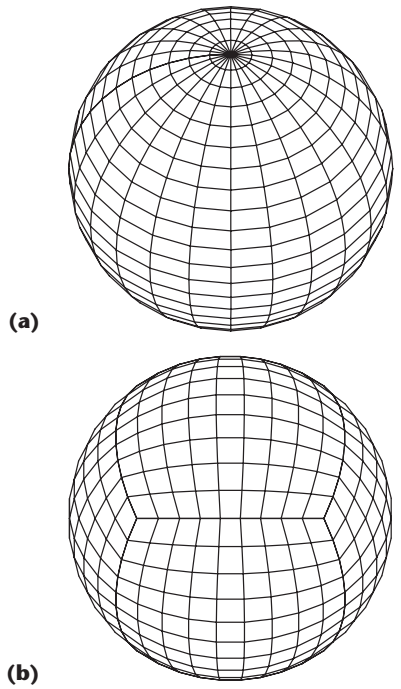
1 Schematic overview of the (a) preprocessing step and (b) collision test.

1b, which shows the various stages during a collision test, illustrates this principle.

As long as there are intersecting sectors of different objects, the algorithm refines the objects. If the algorithm reaches the highest resolution of an object, the collision test is performed on the object primitives, which results in collision determination.

Spherical sampling

A model is described in spherical coordinates (ϕ, θ, r) with respect to its center, an arbitrarily chosen point. The choice of the center point and its position is important for the sampling process. We prefer a center point, with respect to which the model is star-shaped. But even if such a star exists, it is expensive to determine. Heuristic choices such as the center of an enclosing sphere or the model's mass center are good enough to serve as a sampling center. The result of the sampling is a spherical height field $r(\phi, \theta)$ over the parameter domain $[0, 2\pi] \times [-\pi/2, \pi/2]$, which encloses the whole object.



2 (a) Typical sphere parameterization and (b) an alternative one.

As in this parameterization, all meridians coincide with each other at the poles—an intersection test in a near-pole region would lead to many descending tests and a bad performance. Therefore, we use another sphere parameterization (see Figure 2). It subdivides a sphere into six separate, congruent regions and applies an angle-based parameterization to each side.

A simple projection of a bounding box upon a sphere would be sufficient, but it also has some disadvantages. The projection of a bounding box grid onto a sphere results in patches of an unequal area.[2] This problem can be eased by an angle-based parameterization

$$\mathbf{d}(x,y) = \frac{1}{\sqrt{1+\tan^2\frac{\pi}{4}x+\tan^2\frac{\pi}{4}y}} \begin{pmatrix} \tan\frac{\pi}{4}x \\ \tan\frac{\pi}{4}x \\ 1 \end{pmatrix} \qquad (1)$$

with $(x,y) \in [-1, 1]^2$, which delivers much better results, as Figure 3 shows.

Having discretized and sampled the models at the beginning of the preprocessing step, the resulting height fields have to be transformed to get the lower resolution representations.

Subsampling

We can handle subsampling the six charts of our spherical representation much like subsampling a discretized Cartesian height field.

Using the notation introduced elsewhere[6] (and briefly summarized in the "Wavelets" sidebar), we can describe the decomposition by

$$c_k^{j-1} = \max\left(c_{2k}^j, c_{2k+1}^j\right)$$

and

$$d_k^{j-1} = c_{2k}^j - c_{2k+1}^j$$

using approximation coefficients c_k^j and the corresponding detail coefficients d_k^j. We calculate the composition step by

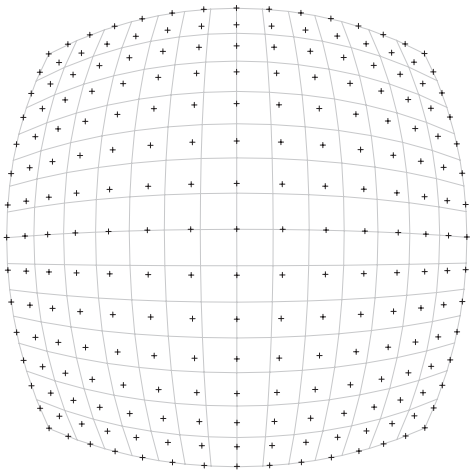
$$c_{2k}^j = c_k^{j-1} + \min\left(0, d_k^{j-1}\right)$$

and

$$c_{2k+1}^j = c_k^{j-1} - \max\left(0, d_k^{j-1}\right)$$

This filter guarantees the required inclusion monotony with an increasing number of detail coefficients. The extensive use of the minimum and maximum functions inspired us to use the max-plus algebra R_{max} (described in Akian et al.[7]), which is an idempotent semi-ring over the set $R \cup \{-\infty\}$ endowed with an addition $\oplus$ and a multiplication $\otimes$ calculated by

$$a \oplus b := \max(a, b) \text{ and } a \otimes b := a + b, \forall a, b \in R$$



3 Distorted, cube projection-based (crosses) and an angle-based (grid) parameterization.

respectively,

$$a \oplus -\infty := a \text{ and } a \otimes -\infty := -\infty \; \forall \, a, b \in R \cup \{-\infty\}$$

This algebra R_{max} is equipped with a commutative, asso-

Wavelets

The starting point for the mathematical framework of multiresolution analysis is a nested set of vector spaces:

$$V^0 \subset V^1 \subset V^2 \subset \ldots$$

As j increases the dimension of V^j, and as a consequence thereof the power to represent a function, increases. The basis functions for the space V^j are known as scaling functions.

The orthogonal complement of V^j in V^{j+1} is called W^j and contains all functions in V^{j+1} that are orthogonal to all those in V^j according to a chosen inner product. The basis functions of W^j are called wavelets.

To keep track of the different scaling functions ϕ_i^j of V^j and the wavelets ψ_i^j of W^j, we combine them into matrices:

$$\Phi^j := \left[\phi_0^j \ldots \phi_{\dim(V^j)-1}^j\right], \Psi^j := \left[\Psi_0^j \ldots \Psi_{\dim(W^j)-1}^j\right]$$

As the subspaces V^j are nested, the scaling functions are refinable—that is, for all $j = 1, 2, \ldots$, there exists a matrix $\boldsymbol{P}^j$ such that $\Phi^{j-1}(x) = \Phi^j(x)P^j$.

Because the wavelet space W^{j-1} is also a subspace of V^j, a matrix $\boldsymbol{Q}^j$ exists satisfying $\Psi^{j-1}(x) = \Phi^j(x)Q^j$. In block matrix notation this can be expressed by

$$\left[\Phi^{j-1} \middle| \Psi^{j-1}\right] = \Phi^j \left[P^j \middle| Q^j\right]$$

We can express a function in some approximation space V^j by its coefficients c_i^j in terms of a scaling function basis:

$$C^j := \left[c_0^j \ldots c_{\dim(V^j)-1}^j\right]$$

A low-resolution version C^{j-1} with a smaller number of coefficients can be created by downsampling—a filtering process describable by a matrix equation $C^{j-1} = A^j C^j$. For many choices of A^j, it is possible to capture the lost detail as a matrix $\boldsymbol{D}^{j-1}$, computed by $\boldsymbol{D}^{j-1} = B^j C^j$. If A^j and B^j (the so-called analysis filters) are chosen appropriately, C^j can be recovered from C^{j-1} and D^{j-1} using P^j and Q^j, which are called synthesis filters:

$$C^j = P^j C^{j-1} + Q^j D^{j-1}$$

As the matrices $\boldsymbol{A}^j$ and $\boldsymbol{B}^j$ satisfy the relation

$$\left[\Phi^{j-1} \middle| \Psi^{j-1}\right] \left[\frac{A^j}{B^j}\right] = \Phi^j$$

it follows

$$\left[\frac{A^j}{B^j}\right] = \left[P^j \middle| Q^j\right]^{-1}$$

Wavelet bases have important applications for signal processing in the vector spaces $\mathbf{R}^n$. Two main approaches for extension from 1D signals to 2D signals in $\mathbf{R}^{n \times m}$ exist: the standard and the nonstandard approach. The standard approach decomposes $\mathbf{R}^{n \times m}$ into $(\mathbf{R}^n)^m$ and $(\mathbf{R}^m)^n$, whereas the nonstandard approach uses rectangular subregions.

ciative, and an idempotent sum and a commutative and associative product.

We can rewrite many parts of the filter bank process in an abbreviated form. For example, we can rewrite the decomposition step as

$$c_k^{j-1} = c_{2k}^j \oplus c_{2k+1}^j$$

so that the analysis filter A^j is a simple matrix. Unfortunately, the analysis filter B^j, which is based on the equation

$$d_k^{j-1} = c_{2k}^j \otimes -c_{2k+1}^j$$

is not of linear form. Of course, the synthesis filters P^j and Q^j exist, otherwise, the reconstruction of the original model would not be possible.

Intersection test

The most relevant part concerning performance is the intersection test routine. The test must report an intersection, if one exists. However, if there is no intersection, the test might report one by mistake. The less faulty, positive results that are reported, the faster the algorithm works, as unnecessary refinements are omitted. Therefore, balancing accuracy and efficiency is essential.

At level zero, both objects to test are enclosed by tight spheres. The test—whether two spheres intersect each other—requires no simplification. But at all other levels of detail, the algorithm has to check two sphere sections, which it analyzes using three methods.

- So-called bounding-volume tests enclose the objects to test within geometrically simpler objects. For example, a polygonal frustum might be used to enclose a sphere section. Although the intersection test of two polygonal frustums is rather simple, the number of tests increases and the enclosing quality is rather bad.
- A totally different approach to check for intersections uses interval/affine arithmetics.[8] In theory, this approach offers an exact and fast intersection test. Although such a test might exist, a practical test has not been implemented yet.
- The intersection test presented elsewhere[3,4] for spherical shells distinguishes the configurations shown in Figure 4. For each configuration, the authors derive an algebraic test for intersection. This test is suitable for our purposes, but due to its complexity and computational expense, we prefer a computationally cheaper alternative.

The following geometrical test shows reasonable performance. The objects to intersect are considered as cones as previously described with a center C, a normalized axis $\mathbf{a}$, a length u, and an opening radius r measured at height $C + \mathbf{a}$.

If the objects, on which the intersection tests are performed, are not considered as volumetric objects but as surfaces, they are handled as truncated cones of which the cone end is cut off at height l, $0 \le l \le u$. In the following, we distinguish both cones and their cone parameters by indices 1 and 2.

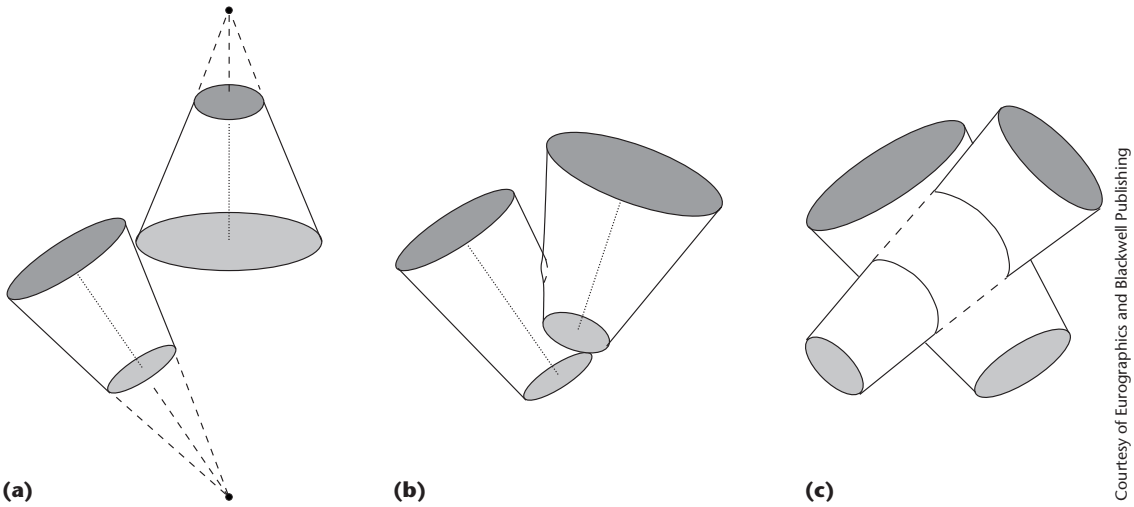
Subject to the orientation of the cone axes, we analyze two cases. The first case deals with nonparallel axes $\mathbf{a}_1$, $\mathbf{a}_2$; the second one deals with parallel lines. We use a heuristically chosen threshold α_{max} to distinguish between the nonparallel and the parallel case:

$$\min(\angle(\mathbf{a}_1, \mathbf{a}_2), \angle(-\mathbf{a}_1, \mathbf{a}_2)) < \alpha_{max}$$

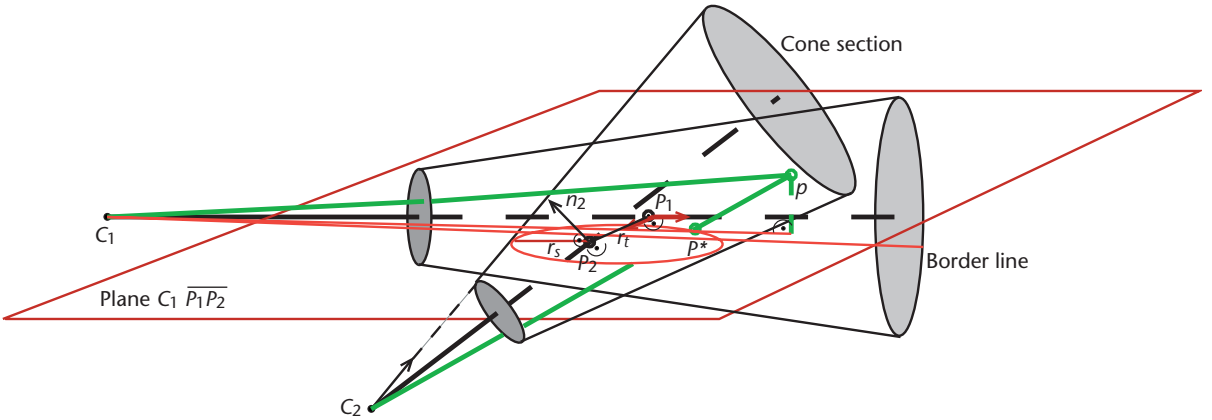
Case 1: Nonparallel axes

In the nonparallel case for both axes, we will determine the shortest distance and the respective perpendicular points P_1, P_2. Let s and t be the corresponding parameters on the axis lines. If a perpendicular point lies outside the clipped cone, it is moved toward the cone. The algorithm computes the other point as the nearest point on the axis of the other cone. In the special case where both perpendicular points are outside, we move the one that is nearest to its clipped cone.

As a quick rejection test we consider the cones as cylinders—that is, if the distance between P_1 and P_2 is greater than the sum of the maximum radii ($u_1 \cdot r_1 + u_2 \cdot r_2$) of both cones, then an intersection is impossible. Next, for a quick acceptance test we consider the spheres with center P_1 respectively P_2 and radius equal to the cone radius there—that is, if the distance between P_1



4 Spherical shell configurations proposed elsewhere.[3,4] (a) Shells intersecting at specific radii, including extreme radii (b–c) and intersecting shells, but not at extreme radii.



5 Cone section in plane $C_1\overline{P_1P_2}$ with additional points p and p^* used in the "Sketch of the Correctness Proof" sidebar. The cone section in the second cone is tested against the border line of the first cone.

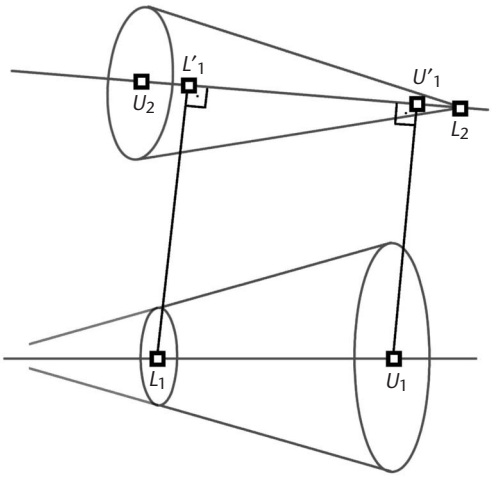
and P_2 is smaller than the sum of sphere radii ($s \cdot r_1 + t \cdot r_2$), then we have an intersection.

If neither a quick rejection nor a quick acceptance occurs, the algorithm performs an exact test based on a cone section. The second cone is intersected with a plane containing the first cone's center C_1 and the line through P_1 and P_2 as illustrated in Figure 5.

This results in a well-known cone section. The first cone is reduced to a line that passes its center C_1 and the point that you get by translating P_1 within the considered plane toward the cone section's focus by length $s \cdot r_1$. The intersection test is then reduced to a 2D intersection test between a cone section and a line. The cone section of a hyperbola cannot occur here, as the considered plane always has an intersection with the other cone's axis by construction. See the "Sketch of the Correctness Proof" sidebar for more information.

Case 2: Parallel axes

In this case, the algorithm analyzes the projection of one cone onto the other cone's axis, as sketched in Figure 6, and vice versa. For each projection, two distance checks of points against their according radii are performed, which results in a total of four checks. The special case of parallel lines is handled separately for optimization—in this case, the whole test can be done by some interval checks, which are much faster.



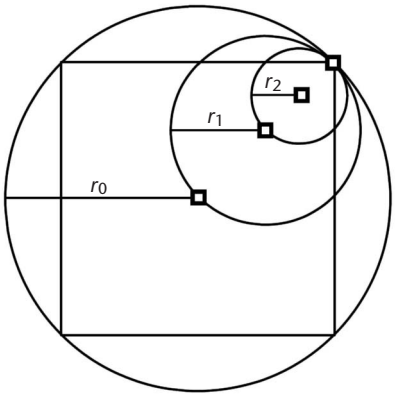
6 Projection of one cone onto the axis of another cone for parallel or near-parallel cone axes.

Although the intersection test might give the impression that it takes a lot of time, it only moderately does so. Due to the fact that the used spherical representation allows for easy rotation and translation, the intersection test can keep up with, for example, discrete orientation polytopes (k-DOPs), if the tested objects are in arbitrary orientation to each other.

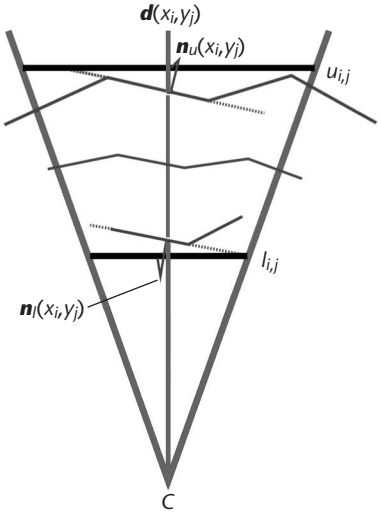
Sketch of the Correctness Proof

Consider both perpendicular points P_1 and P_2 on the cone axes. Let us first study the special cases where one or two points lie outside their clipped cones. In all these special cases with an intersection, there is an intersection between two spheres where one sphere center is located at the lower or upper end on its cone axis. If one point lies outside, then the sphere center can be found by clipping this point. In the case where both points P_1 and P_2 lie outside, assume the point P_2 lying further away from its corresponding cone end gives the sphere center. This sphere intersects the other truncated cone in a sphere between the nearest point P_1 and its nearest cone end. So it suffices to always move the point P_1, which is nearest to its clipped cone.

In the following, we assume that both perpendicular points P_1 and P_2 are located inside their corresponding clipped cones. Let p be a common point of both cones and without loss of generality, assume that p is at the border of cone 2. Let p^* be the intersection point of the line through C_2 and p with the considered plane defined by C_1 and P_1P_2. As point p is at the border of cone 2, the intersection point p^* is at the border of the cone section, which is an ellipse or a parabola. On the side of cone 1, project the line through C_1 and p onto the considered plane. Because p lies inside of cone 1, the projected line lies to the same side of the border line of cone 1. In this way, we have constructed a common point p^* of the cone section inside cone 2 and the half plane of cone 1.



7 Construction for the cone radii r_n at different levels n with $n = 0, \ldots, m$. The diagonal of the parameter square is subdivided n times.



8 Sampling a sphere sector by ray casting.

Implementation

We implemented our new collision-detection algorithm for arbitrary models in the OpenSG scene graph system, where the general model representation is a polygon soup.

Preprocessing

First we choose the center point C of the model as its center of mass. For the model sampling, described in the section "Spherical sampling," two arrays $l_{i,j}$, and $u_{i,j}$ of power-of-two resolution $(i, j = 0, \ldots, 2^m - 1)$, are filled for each of the six sides.

The first array l contains the minimum distance values of model points, whereas the second array u contains the maximum distance values. The entries $l_{i,j}$, $u_{i,j}$ represent the distance for model points in the infinite cone

$$(C, \mathbf{d}(x_i, y_j), r_m)$$

with axis

$$\mathbf{d}\left(x_i, y_j\right), x_i, y_j = -\frac{2^{m-1}-0.5}{2^{m-1}}, \ldots, \frac{2^{m-1}-0.5}{2^{m-1}}$$

and opening $\tan(r_m)$—see Equation 1.

Both the opening angles and the corresponding circle radii r_n for cones at level $n, (n = 0, \ldots, m)$, containing rectangular subparts (see Figure 7), can also be derived from Equation 1.

We can compute these arrays in two different ways. First, we can use ray casting. For each direction $\mathbf{d}(x_i, y_j)$ with domain parameter x_i, y_j, we cast two rays (see Figure 8). The first ray is toward the center point—that is, with origin $C + \Delta \cdot \mathbf{d}(x_i, y_j)$ using a sufficiently large Δ, so that the ray origin lies outside the model, and direction $-\mathbf{d}(x_i, y_j)$. The second ray is from the center point outward. We continue casting rays outward to build a list of faces, pierced by the sampling direction.

During model preprocessing we collect face neighborhoods and store these as a mesh with at least partial connectivity. We do not require any special topology here. For each intersected face we visit neighboring faces until the outer face points lie outside the sampling cone. From the nearest and farthest model layers we determine the minimum and maximum distances $l(i,j)$ respectively $u(i, j)$. The list of triangles is sorted in descending order according to the triangle distance. Additionally, we store with each triangle the diameter of a bounding circle.

In the second approach, we rasterize the model triangles onto the six sides, rather than sampling the model with rays. First, we calculate the side k, $k \in 1, \ldots, 6$ for each point P_i of the triangle (P^1, P^2, P^3). Then we rasterize the triangle in the parameter domain (θ, ϕ)—for side k—with a classical scanline algorithm along the coordinate ϕ. The parameters of the triangle point P^i are

$$\theta^i = \arctan\left((P^i - C)_{k+1 \bmod 3} / (P^i - C)_{k \bmod 3}\right)$$
$$\phi^i = \arctan\left((P^i - C)_{k+2 \bmod 3} / (P^i - C)_{k \bmod 3}\right)$$

For each point met by the scanline algorithm, the distance value

$$d(\theta, \phi) = d((C, (\tan(\theta), \tan(\phi), 1)), (P^1, P^2, P^3))$$

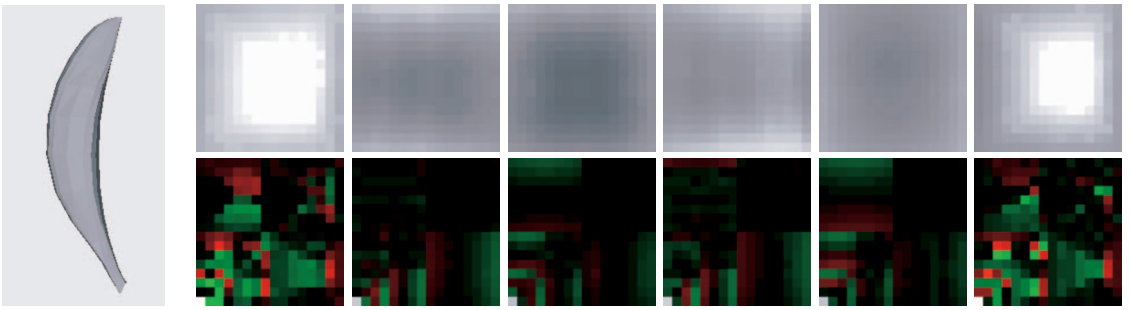
has to be calculated. We can interpolate the distances $d^i = |P^i - C|$ of the triangle points into point (θ, ϕ) by the spherical barycentric coordinates b^1, b^2, b^3 of this point.[9] The distance $d(\theta, \phi)$ is

$$\frac{1}{d\left(\theta,\phi\right)} = b^1 \cdot \frac{1}{d^1} + b^2 \cdot \frac{1}{d^2} + b^3 \cdot \frac{1}{d^3}$$

With these values, we then update the minimum and maximum distance values in the two arrays.

Figure 9 shows the result of model projection onto the six sides. Here, we calculated the distances by ray casting. The ray-casting approach is usually faster than rasterizing the triangles onto the six sides, although it can miss small model parts (of projected size corresponding to a leaf cone). For the rasterization, large triangles sticking out of a side have to be clipped beforehand.

After initialization, the model representation (6×2 distance arrays) is transformed as we described in the "Subsampling" section. To extend the 1D transform into a 2D transform, we use the nonstandard approach,[6] as it generates samples with a constant aspect ratio 1 in all resolutions. This approach applies a 1D transform to the



9 The banana model with its maximum distances on the six sides shown as gray-scale images (with 16×16 pixel resolution) along with the corresponding transforms. The lower left sample contains the overall maximum distance. The other samples are differences as in a nonstandard decomposition, where red and green encode the difference sign (negative and positive), and intensity encodes the difference magnitude (consequently, small differences are black).

array rows, followed by a 1D transform on the array columns.

Figure 9 additionally shows the result of the 2D transform applied to the six sides. As expected, the detail coefficients located in the upper right part of each square and which represent the higher frequencies tend to zero for smooth surfaces with only small variations. The transformation performs in this aspect just like a Haar wavelet transform.

Runtime test

Given the transform data, we can implement the collision test for two objects. Each side $i, (i = 1, \ldots, 6)$, of the first object is tested for intersection against each side j, $(j = 1, \ldots, 6)$, of the second object.

The method, checkIntersect (unsigned i, unsigned j), maintains a queue of spherical shell pairs

$$((C_1, \mathbf{d}(x_1, y_1), l_1, u_1), \quad (C_2, \mathbf{d}(x_2, y_2), l_2, u_2))$$

to test for intersection. This test processes the queue in a breadth-first manner. If the intersection test for the front element of the queue is true, then one of the two cones is refined into four subcones. In our current implementation, we alternate the refinement of cones for the first model with those for the second model. When all cone pairs of this round are tested and the queue is still not empty, then we do one reconstruction step on the corresponding distance arrays. At this time, the cone pairs in the queue build a subset, where each test partner comes from the same refinement level in the corresponding distance arrays.

Collision determination

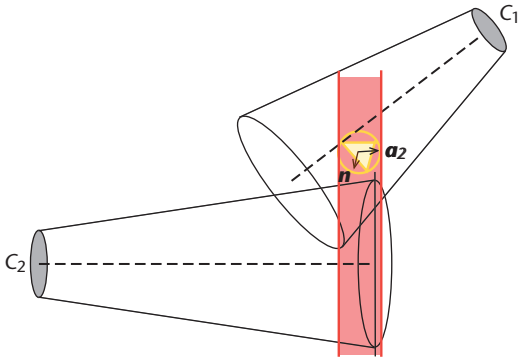
If we reach the leaf levels for both models, then we have to consider the lists of contained triangles for collision determination. In the simplest case of a star-shaped model, we can report an intersection and return one of the contained triangles.

In the general case, all triangle pairs have to be checked for intersection. This can be optimized by using the bounding circle of the first triangle to prune the sorted list of inside triangles of the other model (see Figure 10). During the intersection tests from different, adjacent shells, the amount of overlap between bounding circles should be as small as possible. This avoids repeating triangle intersection tests and potential duplicates in the collision result, which is important for the overall performance. Eliminating duplicates in the collision result afterward is expensive (we did not perform this task in our experiments discussed in the next section).

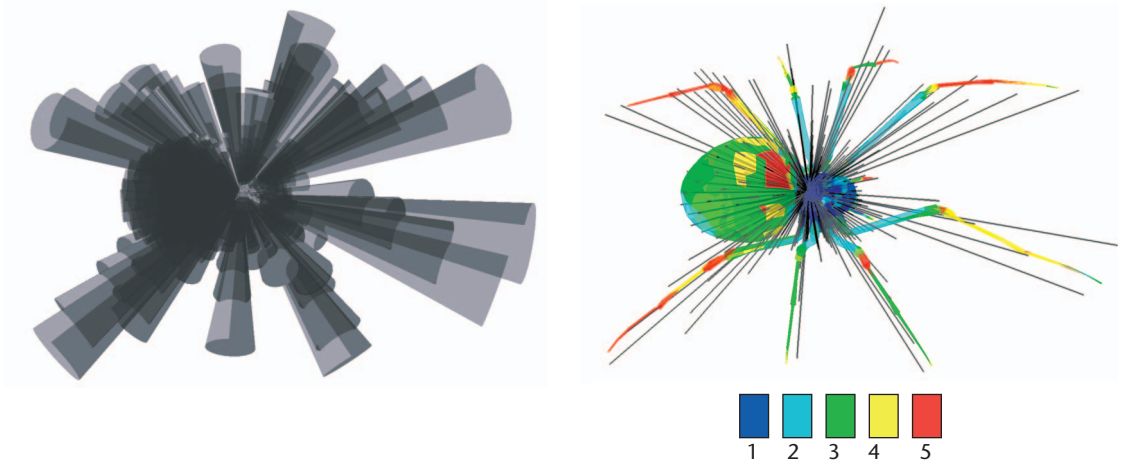
As an intermediate approach we can also report a single triangle per layer of each model. The model layers are immediately defined by the ray-casting precomputation. Each time we continue ray-casting, the layer number is increased by one. But the model can also be classified into layers in a separate pass by comparing triangle distance differences with a given threshold value. The threshold value has to be chosen according to model parameters (see Figure 11 on the next page) for the result with the average triangle diameter as the threshold value. Therefore, we can customize the collision determination according to the application's requirements.

Results

To show the new collision-detection method's efficiency, we ran a series of benchmarks on a Windows PC with a Pentium-M 1.6-GHz processor. Benchmarking collision-detection algorithms realistically is a difficult task. This is because there is a wealth of models with different characteristics and motions relative to each other. Zachmann proposes a benchmarking scheme for two models each contained in a unit box, where the second object performs a number of full-z rotations (in



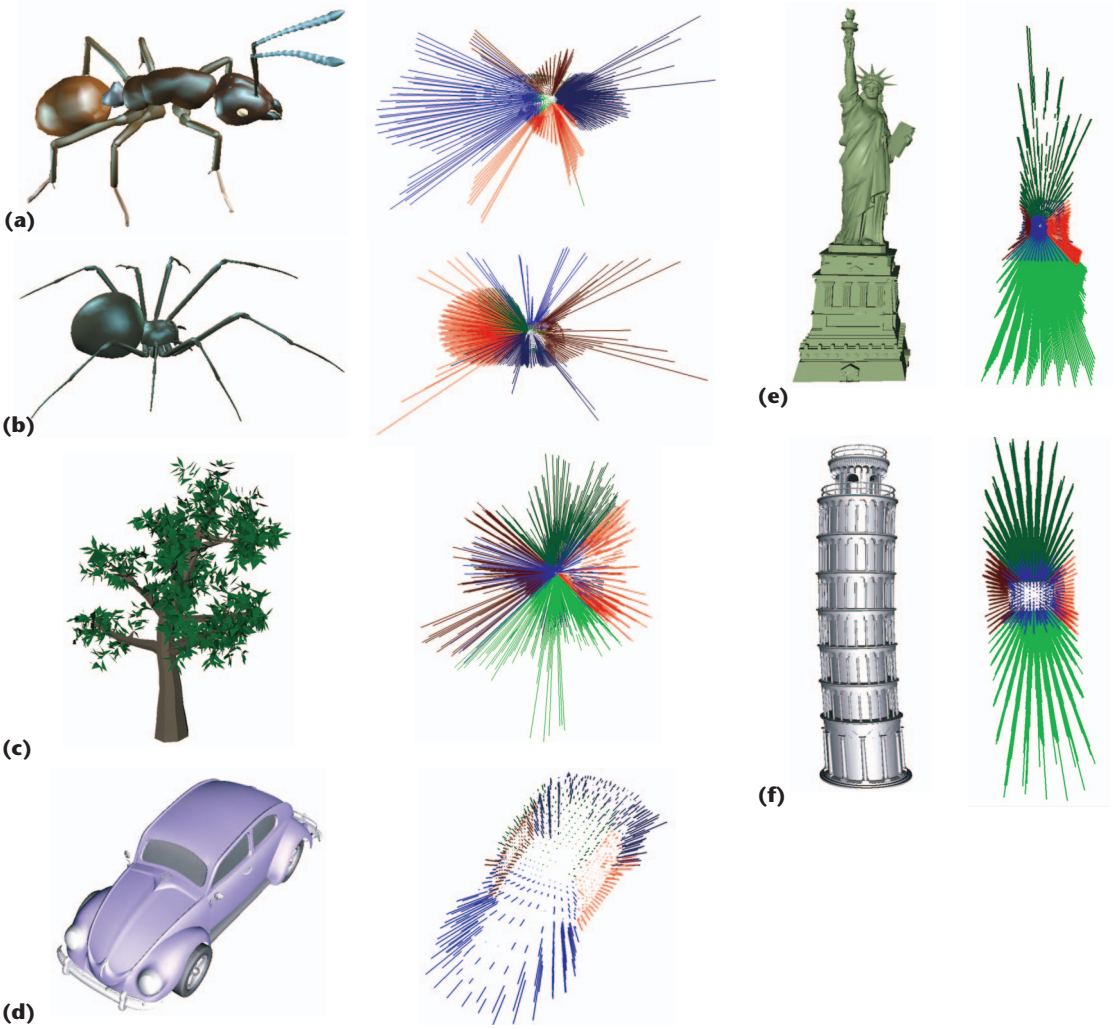
10 Pruning of the cone C_2 for one triangle in cone C_1. The triangle's bounding circle is projected onto the cone axis a_2.



11 Spherical shells for a spider model. The right image part shows the classification of the model triangles into layers. The coloring gives the layer number.

1,000 frames) at decreasing distances relative to the first one.[10]

Throughout all benchmarks, we used ray casting for model sampling with 8×8 and 16×16 samples per side. For collision determination, each sample uses a sorted list of inside model triangles (with their bounding circle radii), and the information subdividing the list into layers. The benchmark *TransformSingle* reports a single triangle pair in a colliding shell pair, *TransformSingle Layer* reports a single triangle pair per layer of a colliding shell pair, and *Transform* reports all triangle pairs in collision (without elimination of duplicates). To compare this approach with well-known collision-detection methods, we included timings using 18-DOP bounding



12 Example models we used in our tests. The models have the following number of triangles: (a) ant (6,667), (b) spider (5,162), (c) tree (4,316), (d) Volkswagen Beetle (57,243), (e) Statue of Liberty (42,225), and (f) Tower of Pisa (153,495).

volumes[10] and oriented bounding volumes as in the publicly available library Rapid (Robust and Accurate Polygon Interference Detection; see http://www.cs.unc.edu/~geom/OBB/OBBT.html). These approaches also report all triangle pairs in collision.

Figures 12a and 12b show the collision test for the models ant and spider. Both models consist of a rather small amount of triangles, but they are highly nonconvex and non-star-shaped.

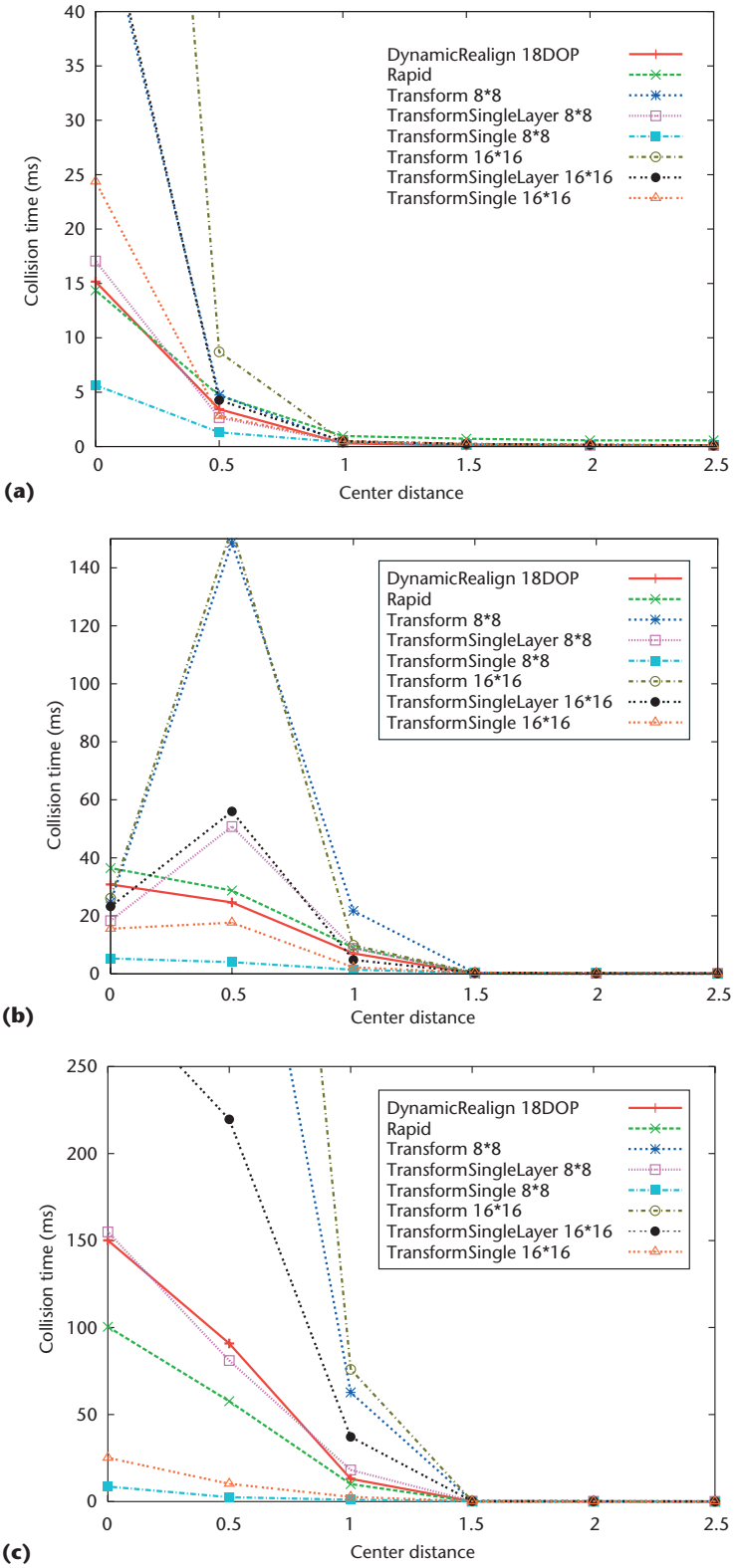
The model ant consists of 1.7 layers per shell on average, and the spider consists of 1.8 layers on average. In the lower left part of the figure, the model sampling is sketched by the clipped axes of the spherical shells. For the triangle sizes in these models, the approximation with 8×8 samples per side is already sufficient. Up to the center distance of 1 unit, the few collision situations can be verified with spherical shells only. The timings are comparable to hierarchies of 18-DOP bounding volumes and slightly better than oriented bounding boxes in this case (see Figure 13a).

Figures 12c and 12d contain the collision test results for a tree model (1.9 layers per shell on average) colliding with a Volkswagen Beetle car model (1.6 layers per shell on average). This test demonstrates the algorithm's ability to handle all model types. Even absolutely unstructured polygon soups, as used for the leaves within the tree model, can be handled the same way as with all other types of representations without any problems.

The last benchmark series performs a collision test between the Statue of Liberty (7.8 layers per shell on average; see Figure 12e) model and the Tower of Pisa (8.4 layers per shell on average; see Figure 12f). These results should be compared with those for the tree/Beetle model. If reporting just a single triangle per spherical shell, the timings do not vary much, although the model complexity is much higher. In this mode of collision determination, collision times are independent of the triangle count. Collision time depends only on the sampling density and the volume between the inner and outer bounding shell.

The models have their largest extent along the *y*-axis but none of the bounding volume hierarchy approaches can take advantage of this in the close-proximity situation (center distance 0). For center distance 0.5, there are fewer spherical shell tests and more triangle to triangle tests compared to center distance 0. It is interesting that the transition from center distance 0.5 to 0 can be used to assess the sampling density.

For the triangle sizes in the models tree and Beetle the approximation with 16×16 samples per side is sufficient. For the small triangle sizes in the models Liberty and Tower of Pisa and testing all triangle pairs, the approximation with 8×8, 16×16 samples per side is not sufficient, and 32×32 samples still gives better results.



13 Corresponding test timings of models in Figure 12: (a) ant and spider, (b) tree and Volkswagen Beetle, and (c) Statue of Liberty and Tower of Pisa.

Future work

Our collision-detection algorithm is simple to implement and offers a storage scheme that consumes half the space compared to the full storage of the hierarchical spherical distance field.

A field of future work is to adapt this technique to deformable models. In its current form, with regular sampling, it's difficult to update the lists of inside triangles used for the advanced collision determination. Recomputing the outer and inner bounding shell is possible by graphics hardware in two passes. As the depth

buffer of OpenGL receives distances to the eye plane, we will need an additional warping into eye point distances. Furthermore, a GPU implementation of the spherical shell intersection test seems possible. The combination of both approaches will realize a full GPU implementation of collision detection for the outer shells of objects. ■

Acknowledgments

We thank Volkswagen AG, Division of Virtual Prototyping, for providing us with the Beetle and New Beetle models. Special thanks go to the reviewers for some valuable comments and improvements. We acknowledge the support by the German Federal Ministry of Education and Science (bmb+f) in the OpenSG PLUS project.

References

1. L. Lin and D. Manocha, "Collision and Proximity Queries," *Handbook of Discrete and Computational Geometry*, 2nd ed., J.E. Goodman and J. O'Rourke, eds., CRC Press, 2004.
2. E. Praun and H. Hoppe, "Spherical Parametrization and Remeshing," *ACM Trans. Graphics*, vol. 22, no. 3, 2003, pp. 340-349.
3. S. Krishnan et al., "Spherical Shells: A Higher-Order Bounding Volume for Fast Proximity Queries," *Proc. IEEE 3rd Int'l Workshop Algorithmic Foundations of Robotics*, P.K. Agrawal, L.E. Kavraki, and M. Mason, eds., A K Peters, 1998, pp. 177-190.
4. S. Krishnan et al., "Rapid and Accurate Contact Determination between Spline Models Using ShellTrees," *Computer Graphics Forum*, vol. 17, no. 3, 1998, pp. 315-326.
5. G. Hamlin, R. Kelley, and J. Tornero, "Efficient Distance Calculation Using the Spherically-Extended Polytope (s-tope) Model," *Proc. IEEE Int'l Conf. Robotics and Automation*, IEEE CS Press, 1992, pp. 2502-2507.
6. E.J. Stollnitz, T.D. DeRose, and D.H. Salesin, *Wavelets for Computer Graphics: Theory and Applications*, Morgan Kaufmann, 1996.
7. M. Akian et al., "Linear Systems in (max,+) Algebra," *Proc. 29th Conf. Decision and Control*, IEEE CS Press, 1990, pp. 151-156.
8. K. Bühler, "Taylor Models and Affine Arithmetics—Towards a More Sophisticated Use of Reliable Arithmetics in Computer Graphics," *Proc. 17th Spring Conf. Computer Graphics* (SCCG), IEEE CS Press, 2001, pp. 40-48.
9. S. Morigi and G. Casciola, "Inverse Spherical Surfaces," *J. Computational and Applied Mathematics,* vol. 176, no. 2, 2005, pp. 411-424.
10. G. Zachmann, "Rapid Collision Detection by Dynamically Aligned DOP-Trees," *Proc. Virtual Reality Ann. Int'l Symp.*, IEEE CS Press, 1998, p. 90.

Christoph Fünfzig *is pursuing a PhD in computer science at Braunschweig University of Technology, Germany. His research interests include practical computational geometry and animation and simulation in computer graphics. Fünfzig has an MS in computer science from the University of Bonn, Germany. Contact him at c.fuenfzig@tu-bs.de.*

Torsten Ullrich *is pursuing a PhD in computer science at Graz University of Technology, Austria. His research interests include modeling and reconstruction in computer-aided geometric design. Ullrich has an MS in mathematics from the University of Technology in Karlsruhe, Germany. Contact him at t.ullrich@cgv.tugraz.at.*

Dieter W. Fellner *is director of the Institute of Computer Graphics and Knowledge Visualization and a professor of computer science at Graz University of Technology, Austria. His research interests include computer graphics, modeling, immersive systems, and graphics in digital libraries. Fellner has an MS and a PhD from Graz University of Technology, Austria. He is a member of the IEEE, ACM, Eurographics, and the German Computer Society, and is on the editorial board of* IEEE Computer Graphics and Applications. *Contact him at d.fellner@tugraz.at.*