Fast and Simple 2D Geometric Proximity Queries Using Graphics Hardware

Kenneth E. Hoff III, Andrew Zaferakis, Ming Lin, Dinesh Manocha

Department of Computer Science
University of North Carolina at Chapel Hill
http://www.cs.unc.edu/~geom/PIVOT/

ABSTRACT

We present a new approach for computing generalized proximity information of arbitrary 2D objects using graphics hardware. Using multi-pass rendering techniques and accelerated distance computation, our algorithm performs proximity queries not only for detecting collisions, but also for computing intersections, separation distance, penetration depth, and contact points and normals. Our hybrid geometry and image-based approach balances computation between the CPU and graphics subsystems. Geometric object-space techniques coarsely localize potential intersection regions or closest features between two objects, and image-space techniques compute the low-level proximity information in these regions. Most of the proximity information is derived from a distance field computed using graphics hardware. We demonstrate the performance in collision response computation for rigid and deformable body dynamics simulations. Our approach provides proximity information at interactive rates for a variety of simulation strategies for both backtracking and penalty-based collision responses.

Keywords: Proximity queries, collision detection, penetration depth, graphics hardware acceleration, multi-pass techniques.

1 INTRODUCTION

Many applications of computer graphics or computer simulated environments require spatial or proximity relationships between objects. In particular, dynamic simulation, haptic rendering, surgical simulation, robot motion planning, virtual prototyping, and computer games often require many different proximity queries simultaneously at interactive rates. We focus on interactive computation of the following proximity queries between 2D objects: collision detection, intersection, minimum separation distance, penetration depth, and contact points and normals.

Algorithms for determining collisions, intersections, and minimum separation distances have been extensively researched. Many are restricted to convex objects [2,4,6,16] or are based on hierarchical bounding-volume or spatial data structures that require considerable precomputation and are best suited for rigid geometry [8,12,14,19]. Some algorithms handle dynamically deforming geometry by either having prior knowledge of motion trajectories [22] or by using very specialized algorithms [1]. In our approach, we emphasize the handling of non-convex, dynamically deformable objects with no precomputation or knowledge of object motions.

Penetration depth is typically defined as the minimum translational distance needed to separate two objects. We define it with respect to a point as the minimum translational distance and direction needed to separate a penetrating point from an object's interior. This information is useful for penalty-based collision response computation. Dobkin et al. have presented an algorithm to compute the intersection depth of convex polytopes, though no practical implementation is known [3]. In general, no robust and efficient algorithms are known for computing the penetration depth and direction for general, non-convex primitives.

Our algorithm relies on the computation of discretized distance fields and graphics hardware-accelerated geometric computation. Distance fields – scalar fields that specify minimum distance to a shape for all points in the field – have been used for many applications in graphics, robotics and manufacturing [5,9]. Common algorithms for distance field computation are based on level sets [21] or adaptive techniques [5]. However, they either require static geometry, extensive preprocessing, or lack tight error bounds. Graphics hardware has been used to accelerate a number of geometric computations, such as visualization of constructive solid geometry models [7] and cross-sections and interferences [20]. However, these only compute intersections, not distance-related queries. Algorithms also exist for motion planning using graphics hardware acceleration and distance fields [11, 13,15,18]. More recently, an algorithm has been proposed to compute generalized Voronoi diagrams and distance fields using graphics hardware [10]. Its application to motion planning was presented in [11,18].

Our algorithm combines coarse traditional hierarchical approaches and multi-pass rendering techniques with the graphics hardware-accelerated distance field computation presented in [10]. The main features of our approach include a unified framework for all proximity queries, generality to non-convex polygons, no required precomputation or complex data structures, computational efficiency allowing interactive queries on current PCs, robustness requiring no special-case handling of degeneracies, portability across various CPU/graphics combinations, and error-bounds on approximations. We have implemented our algorithm on PC and SGI platforms, and demonstrated its performance in computing collision responses in both rigid- and deformable-body dynamic simulations. Our current algorithm and implementation focuses on 2D polygonal objects, but the basic design principles extend to 3D and are the focus of our current work.

2 OUR APPROACH

While algorithms exist for performing some of the proximity queries in both 2D and 3D, none meet all of our requirements even

I3D '2001, Research Triangle Park, NC USA
© ACM 2001 1-58113-292-1/01/01 ...$5.00

in 2D. Our first step in developing a general unified approach that is efficient and robust in practice focuses on the general 2D proximity problem. Given a collection of 2D objects, we perform coarse geometric localization to find rectangular regions of space that contain either potential intersections or closest feature pairs between objects. We uniformly point-sample these regions and use polygon rasterization hardware to compute object intersections, closest points, and the distance field. The distance field and its gradient vector field provide the distance and direction to the nearest feature for each point in the localized region, which gives the contact normals, minimum separation distances, or penetration depths. Our core algorithm computes the proximity information between two 2D, simple, possibly non-convex polygons. Higher-order primitives are tessellated into polygons with bounded distance deviation error. In our hybrid approach, there are two top-level operations: (1) geometric object-space operations to coarsely localize potential intersection regions or closest features, and (2) image-based operations using graphics hardware to compute the proximity information in the localized regions.

2.1 Geometric Localization

The image-based queries operate on a uniform grid of sample points in regions of space containing potential interactions. The graphics hardware pixel framebuffer is used as the grid and the queries become pixel operations, therefore the performance varies dramatically with the pixel resolution. To avoid excessive load, a geometric localization step is used to window regions of potential interaction or as a trivial rejection stage. This hybrid geometry/image-based approach helps balance the load between the CPU and graphics subsystems, giving us portability between different workstations with varying performance characteristics. Using more sophisticated geometric techniques to tightly localize potential intersections or closest feature pairs dramatically reduces the graphics pipeline overhead, but increases the CPU usage and the complexity of the algorithm. We use coarse bounding-volume hierarchies to achieve this balance between speed and complexity, and CPU and graphics usage.

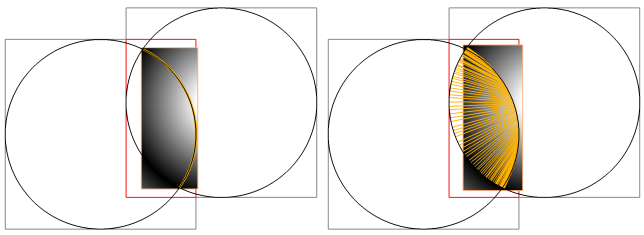

Figure 1: Points on the boundary of left circle intersecting the volume of the right circle, a tight-fitting bounding box around these penetrating points, and the distance field of the right circle computed in this bounding region (left). Gradient vectors at the penetrating points computed using central differencing in the distance field. The lengths represent the distance to the boundary (right). The top-level bounding boxes and their intersection used for computing the intersection points are also shown.

There are many general and efficient algorithms available for localizing geometry based on bounding-volume hierarchies [8,12,14,19]. However, for exact intersection testing these algorithms typically perform well only on static geometry where the hierarchy can be precomputed. In order to handle dynamic deformable geometry with no precomputation, we use coarse levels for efficient trivial rejection and to obtain reasonable geometric localization. In addition, we perform lazy evaluation of relevant portions of the hierarchies while performing the collision or distance query. A subtree rooted at a particular node is only computed if its children need to be visited during the query traversal. The trees are destroyed after every proximity query, and no actual tree data structures are required since the child nodes are recursively passed to the query routine. A maximum height of each object tree is used to balance the CPU and graphics load.

2.2 Image-based Proximity Queries

The proximity queries are simplified using uniform point sampling and accelerated with graphics hardware. This image-based approach helps decouple the objects' geometric complexity from the computational complexity for a specified error tolerance. The geometric localization step improves the performance since large areas of space and portions of the objects can be rejected from the query computation. We point-sample the geometry and the space around the geometry within the localized regions with a uniform rectangular grid and perform the queries on this volumetric representation using graphics hardware acceleration. The image-based queries include computing intersections between objects, computing the distance field of an object boundary, and computing the gradient of the distance field. Variations of these basic operations are used to perform the remaining queries.

2.2.1 Intersections

There are three types of intersections possible between two polygonal objects: boundary-boundary, boundary-volume, and volume-volume (boundary-volume is shown in Figure 1). We render both objects within a localized region using the graphics hardware and treat overwritten pixel sample points as the intersection points. The type of intersection determines whether the boundary or the interior of the object is rendered. Several strategies are given for detecting overwritten pixels (Table 1).

Multi-pass operations for finding object intersections

Buffer	Clear val	Render B	Render A	Intersection
Stencil	0	increment by 1	for all pixels==1, incr by 1	stencil value: 2
Color: blend ops	0,0,0	set color to 128,128,128	in color 127,127,127 with additive blend	color = 255,255,255
Color: logic ops	0,0,0	set color to 127,127,127	in color 128,128,128	color = 255,255,255
Color and Depth	0,0,0 and 1	depth = 0 depth func = always pass	depth = 0 depth func = equals Color = 255,255,255	color = 255,255,255

Table 1: OpenGL multi-pass rendering options for finding the overwritten pixels. The basic ops: a buffer is cleared; object B is rendered setting buffer values of all covered pixels; object A is rendered changing buffer values of pixels covered by A and B; intersection points are represented by pixels whose buffer values are set in the last pass. Each approach varies in performance, in the resulting buffer state, and in the sophistication needed in the underlying hardware implementation.

The error in the intersection calculation is governed by the pixel resolution. Given a distance error bound d, we choose a resolution so that no point in the rectangular region can be farther than d from a pixel sample point (d is the half diagonal length of a pixel grid cell). These error bounds hold for filled polygons, since all pixels in the interior of the polygon will be rasterized. Line segment rasterization does not guarantee that all pixels within d distance of the line will be set, so we draw an offset polygon surrounding the line segment that is d distance away from the line segment using the bounded-error distance mesh presented in [10].

The intersection operation requires clearing a buffer, rendering the objects into the potential intersection region, reading the buffer containing the intersection information, and searching through the image to find the intersection pixels. We avoid the full-screen clear by drawing a polygon the size of the localized region. Hardware min/max or histogram queries eliminate read back and the per-pixel search when no intersections have occurred, but

these operations may not be available on some platforms. In this case, the coarse bounding-volume hierarchy is used to reject object pairs. When the image operations dominate the query time, performance can be improved by increasing the error tolerance or by improving the geometric localization step by traversing deeper levels in the hierarchy. The running time of these image operations is largely independent of the object complexity, thus becoming negligible for complex objects.

The complexity of object rasterization grows linearly with respect to the number of vertices. Computing intersections geometrically between two polygon boundaries is worst case $O(n^2)$ since all edges could intersect. The complexity of our algorithm is O(n) where n is the number of vertices. The hierarchical geometric localization step is also O(n) since the maximum depth of the tree is held constant.

2.2.2 Distance Field

We use a variation of the algorithm described in [10] for constructing generalized Voronoi diagrams using graphics hardware for 2D polygonal objects. This approach computes an image-based representation of the Voronoi diagram in both the color and the depth buffers. A pixel's color identifies the polygon feature (vertex or edge) that is closest to that pixel's sample point; its depth value corresponds to the distance to the nearest feature. The depth buffer is an image-based representation of the distance field of the polygon boundary. The distance field is computed by rendering 3D bounded-error polygonal mesh approximations of the distance function where the depth of the rendered mesh at a particular pixel location corresponds to the distance to the nearest 2D polygon feature. Distance values at arbitrary points are bilinearly interpolated from the four nearest pixel distance values.

The algorithm by Hoff et al. only gives unsigned distance [10]. We need signed distances to avoid problems when computing the gradient near an object boundary for computing surface contact normals. We extend this algorithm to compute signed distances by distinguishing the inside and outside regions of the object using any of the available buffers to encode the "negative" interior of the object. We simply render the polygon, setting values in a pixel buffer. For each distance value, we also have a sign value that is read from this other buffer. Several possibilities include: setting the stencil buffer to 1; setting the color buffer to white; setting the most significant bit of the color ID in the Voronoi computation. For arbitrary points, the sign value can also be bilinearly reconstructed between 0 and 1. Values less than or equal to 0.5 can be positive and values greater than 0.5 can be negative.

Distance field computation requires clearing the depth buffer, rendering the objects' distance mesh, reading back the depth buffer, and rendering and reading of sign values. This is often more efficient than computing the intersection since we only need distance values at the intersection points (or at closest feature or penetrating points). In fact, we may not even need to read back the entire buffer since we could read just the individual pixel locations that we are querying (Figure 1).

2.2.3 Gradient of the Distance Field

We compute the gradient of the distance field at pixel locations by using central differences. For an arbitrary point, we compute the gradient as the bilinear interpolation of the gradients at the four surrounding pixel locations. In practice, this gives reasonable results even with the error and lack of C^1 or higher continuity in the polygonal distance mesh approximations used to compute the distance field (Figure 1). Gradients are computed in software for selected points after reading back the distance values. If the entire gradient field is desired, we could accelerate the computation using multi-pass rendering. For the *x* component of the gradient, we could subtractively blend the distance image shifted two pixels to the left with the original distance image. For the *y* component, we blend with the image shifted two pixels down. The division by 2 is performed by a multiplicative blend of 0.5. Unfortunately, subtractive blending is currently not available on all platforms even though it has been accepted into most graphics APIs, and the limited precision of pixel arithmetic may cause noticeable errors.

2.2.4 Other Proximity Queries

Given the basic operations of computing intersections, distance fields, and gradient of the distance field, we can perform the other proximity queries mentioned in section 1.

Penetration Depth and Direction: For a point on object A that is penetrating object B, we define the penetration depth and direction for the point as the distance and direction to the nearest feature on B. This is given by the distance field and its gradient computed at the penetrating point. Penetrating points are found using the intersection operation.

Contact Points and Normals: Ideally, the contact points are simply the intersections of the object boundaries; however, we often need the set of points that are almost in contact. For a given contact distance threshold *d*, we find all boundary points that are within *d* distance of each other. The basic approach is slightly modified to efficiently handle this query. First of all, in the geometric localization stage we find the potential intersection between the two polygons that are slightly thicker. This is handled by enlarging all bounding boxes by *d* in each. We then find the intersection between the boundaries by drawing the objects' boundary line segments with an enlarged offset of *d* (using the distance mesh from [10]) and finding intersecting pixels. Normals at each contact point are computed from the gradient of the signed distance field (signed distance to avoid distance discontinuity near the object boundary).

Closest Point: We find the point on object A that is closest to object B by rendering the boundary of object A in the localized region of A containing its closest feature, rendering the distance field of B in this same region, and then searching the boundary points of A and finding the point that is closest to B.

Separation Distance and Direction: We find the minimum separation distance and direction between two objects A and B by first computing the closest point on A to B and vice versa. Ideally, we find the closest point on B to A from the distance value and gradient at the closest point on A to B, but the amplification of errors over the greater distance may cause problems. The distance between these two closest points is the separation distance and the line segment between them gives the separation direction.

3 PERFORMANCE

We demonstrate the effectiveness of our proximity queries in computing collision responses for interactive dynamic simulations of rigid and deformable objects. We compute the collision response for a particle and use a collection of particle responses to extend to rigid and deformable bodies [1,19]. We implemented collision responses for simulations with and without penetration constraints. In constrained simulation, penetration is avoided through a backtracking algorithm that finds the state of all objects "just before" a collision. A bisection search in time is performed between the last non-collision state and the collision state for all

objects in the scene, and the collision response is then computed for the objects that are in close contact. In unconstrained simulation, penetration is allowed, but a spring-based restoring penalty force proportional to the penetration depth is applied to the object until separation occurs. Collision responses between object pairs are handled locally without requiring global update of the entire system.

Each collision response requires different proximity information. Constrained simulation requires points of *close* contact and the contact normals. Unconstrained simulation requires points of penetration and their penetration depths. The effectiveness of our approach is most clearly shown in the unconstrained, penalty-based approach because of the difficulty in computing penetration depth. Earlier algorithms give only coarse approximations without error bounds or are only restricted to convex objects.

We tested the system in several different contact scenarios. In each simulation, the user provides the initial position, orientation, and velocity of a collection of objects, and the appropriate collision responses are computed as the simulation advances. See the colorplate for descriptions of the simulations. Each simulation is performed on rigid-bodies except for **wavy**, but the same proximity query algorithm was used on all simulations. More deformable bodies were not shown because of the difficulty in developing effective deformable simulations. In table 2, we show the average total per-frame proximity query times. **Wavy** requires more time because there are large areas of continuous close contact as the shapes conform to each other when colliding. In table 3 we show the effects of the distance error on performance.

Average Total Per-frame Proximity Query Times					
Demo	**Objects**	**Lines**	**GeForce2**	**InfiniteReality2**	**ATI Rage Pro LT**
Map	6	719	0.281ms	0.901ms	0.434ms
Gears	13	391	0.015	0.026	0.064
Links	15	440	0.020	0.052	0.038
Cars	18	266	0.007	0.026	0.015
Wavy	2	200	1.030	2.360	2.990

Table 2: Performance timings for dynamics simulations. The number of objects, number of line segments, and the average total time in milliseconds to run proximity queries on all objects in the scene per frame is reported. Timing data was gathered from three machines: a Pentium-III 933MHz desktop with a 64Mb GeForce2, a SGI 300MHz R12000 with InfiniteReality2 graphics, and a Pentium-III 750Mhz laptop with ATI Rage Pro LT graphics.

Effects of Error Tolerance on Performance of Wavy			
Error	**GeForce2**	**InfiniteReality2**	**ATI Rage Pro LT**
d/4	0.710ms	1.270ms	5.560ms
d/2	0.315	1.000	1.850
d	0.211	0.930	0.895
2d	0.176	0.879	0.631
4d	0.165	0.876	0.535

Table 3: The effect on performance when changing the distance error tolerance *d*. We used proximity queries on the **wavy** demo with no collision response. The error determines the number of pixels used in the image-based operations. Systems with low graphics performance are more directly affected by the choice of *d* (see ATI Rage Pro LT); however, as the error is increased there is less dependence on graphics performance and the faster laptop CPU overtakes the InfiniteReality2 system.

4 CONCLUSION

We have presented a hybrid geometry- and image-based algorithm for computing geometric proximity queries between two arbitrary 2D objects using graphics hardware. This approach has a number of advantages over previous approaches since the unified framework allows us to compute all the queries, including penetration depth and contact normals. Furthermore, it involves no precomputation and handles non-convex polygons; as a result, it is also applicable to dynamic or deformable geometric primitives. In practice, we have found the algorithm to be simple to implement, quite robust, fast (considering the complexity of the queries), and very flexible. We have developed an interactive 2D dynamic simulation system for rigid and deformable objects to illustrate the effectiveness of our approach. We are currently extending this framework to 3D for interactive proximity queries on complex, dynamic geometry.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their helpful suggestions. This work was supported by ARO DAAG55-98-1-0322, DOE ASCII Grant, NSF NSG-9876914, NSF DMI-9900157, NSF IIS-982167, ONR Young Investigator Award, and Intel.

REFERENCES

[1] D. Baraff, *Dynamic Simulation of Non-Penetrating Rigid Bodies*. Ph.D. Thesis, Dep of Comp. Sci., Cornell University, March 1992

[2] S. Cameron, *Enhancing GJK: Computing Minimum and Penetration Distance between Convex Polyhedra*. International Conference on Robotics and Automation, 3112-3117, 1997

[3] D. Dobkin, J. Hershberger, D. Kirkpatrick, S. Suri, *Computing the Intersection Depth of Polyhedra*. Algorithmica, 9(6), 518-533, 1993

[4] S. Ehmann and M. Lin. *Accelerated Proximity Queries Between Convex Polyhedra By Multi-Level Voronoi Marching*. Proc. International Conf. on Intelligent Robots and Systems, 2000

[5] S. Frisken, R. N. Perry, A. P. Rockwood, T. R. Jones, *Adaptively Sampled Distance Fields: A General Representation of Shapes for Computer Graphics*. SIGGRAPH 00, 249-254, July 2000

[6] E. G. Gilbert, D. W. Johnson, S.S. Keerthi. *A Fast Procedure for Computing the Distance Between Objects in Three-Dimensional Space*. IEEE J. Robotics and Automation, RA(4): 193-203, 1988

[7] J. Goldfeather, S. Molnar, G. Turk, and H. Fuchs. *Near Real-time CSG Rendering Using Tree Normalization and Geometric Pruning*. IEEE Computer Graphics and Applications, 9(3):20–28, May 1989

[8] S. Gottschalk, M. C. Lin, D. Manocha, *OBB-Tree: A Hierarchical Structure for Rapid Interference Detection*. SIGGRAPH 96, 171-180, 1996

[9] G. Hirota, S. Fisher, M. Lin. Simulation of Non-penetrating Elastic Bodies Using Distance Fields. University of North Carolina at Chapel Hill Technical Report: TR00-018. Spring 2000

[10] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. *Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware*. SIGGRAPH 99, 277-285, 1999

[11] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. *Interactive Motion Planning Using Hardware-Accelerated Computation of Generalized Voronoi Diagrams*. Proc. of IEEE International Conf. on Robotics and Automation, 2000

[12] P. M. Hubbard, *Interactive Collision Detection*. IEEE Symposium on Research Frontiers in Virtual Reality. 24-31, 1993

[13] R. Kimmel, N. Kiryati, A. Bruckstein, *Multi-Valued Distance Maps for Motion Planning on Surfaces with Moving Obstacles*. IEEE Transactions on Robotics and Automation, vol 14: 427-438, 1998

[14] D. Johnson, E. Cohen, *A Framework for Efficient Minimum Distance Computation*, IEEE Conf. On Robotics and Animation, 3678-3683, 1998

[15] J. Lengyel, M. Reichert, B.R. Donald, and D.P. Greenberg. *Real-time Robot Motion Planning Using Rasterizing Computer Graphics Hardware*. Computer Graphics (SIGGRAPH 90 Proc.), vol. 24, pgs 327–335, Aug 1990

[16] M. Lin, J. Canny. *Efficient Algorithms for Incremental Distance Computation*. IEEE Transactions on Robotics and Automation, 1991

[17] B. Mirtich, *Impulse-Based Dynamic Simulation of Rigid Body Systems*. Ph.D. Thesis, University of California, Berkeley, Dec 1996

[18] C. Pisula, K. Hoff, M. Lin, and D. Manocha. *Randomized Path Planning for a Rigid Body Based on Hardware Accelerated Voronoi Sampling*. Proc. of Workshop on Algorithmic Foundations of Robotics, 2000

[19] S. Quinlan, *Efficient Distance Computation between Non-Convex Objects*. International Conf. on Robotica and Automation, 3324-3329, 1994

[20] J. Rossignac, A. Megahed, and B. Schneider. *Interactive Inspection of Solids: Cross-sections and Interferences*. SIGGRAPH 92, 26, 353–360, July 1992.

[21] J. Sethian, *Level Set Methods*, Cambridge University Press, 1996

[22] J. Snyder, A. Woodbury, K. Fleischer, B. Currin, A. Barr, *Interval Methods for Multi-Point Collisions Between Time Dependent Curved Surfaces*. ACM Computer Graphics, 321-334, 1993

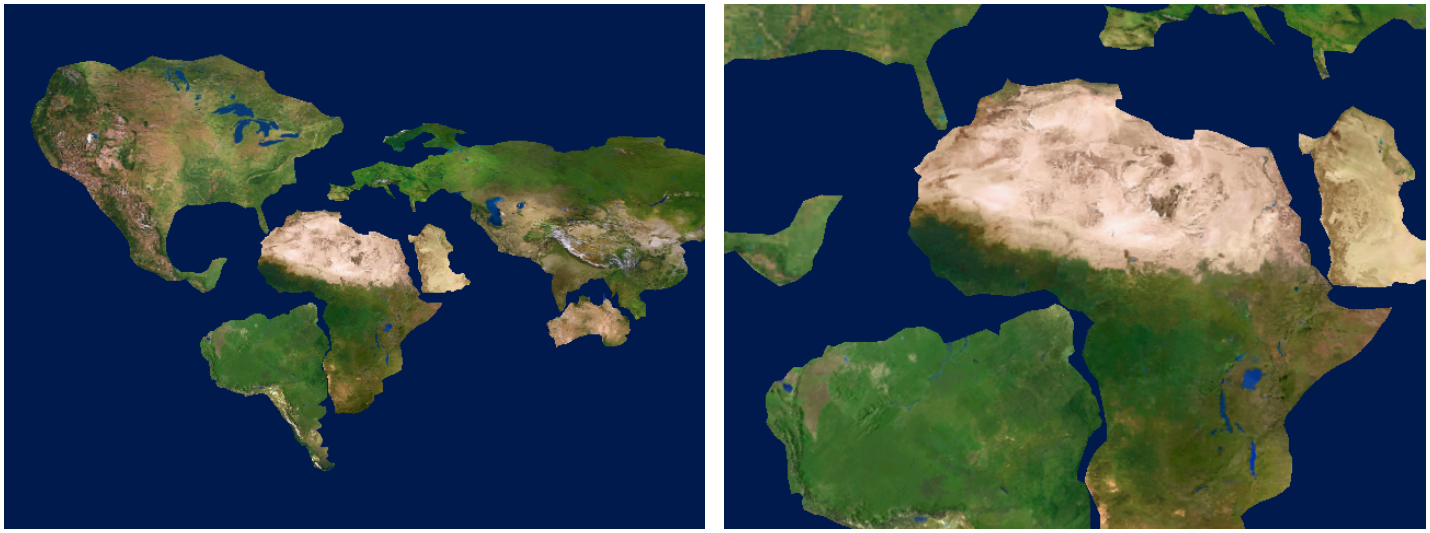

PLATE 1: **Map** (large non-convex objects, frequent simultaneous close contact). Our approach computes proximity query information needed for penalty-based collision response between complex non-convex objects. For each penetrating point, we compute a minimal penetration depth and direction and apply a penalty force to resolve the collision. Intersections between the top-level axis-aligned bounding boxes were used as potential intersection regions. A coarse hierarchical search with oriented bounding boxes would find smaller potential intersection regions, thus improving performance. Even with this simplified search, we achieved interactive performance on several difference machines with widely varying CPU/graphics combinations (see Table 2).

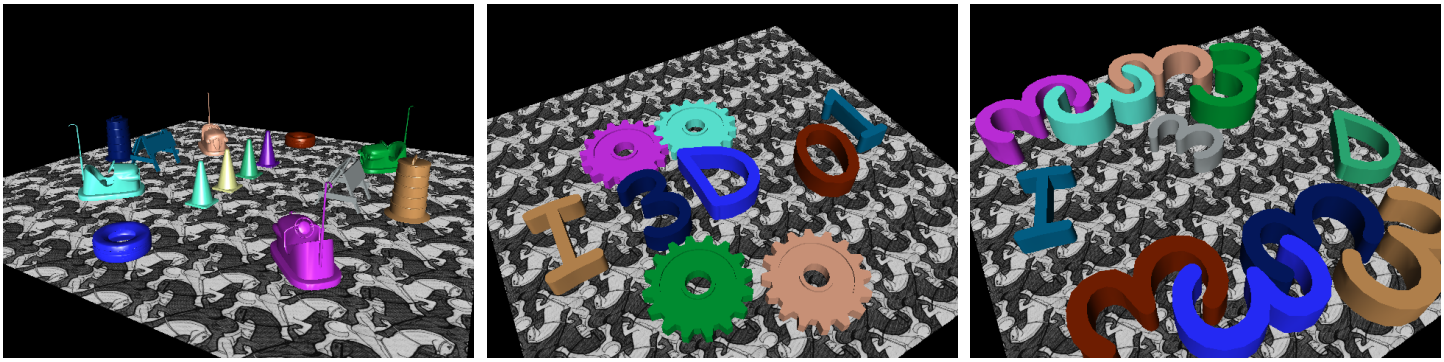

PLATE 2: **Cars** (convex objects, less frequent contact), **Gears** (non-convex, less frequent interlocking contact), and **Links** (non-convex objects, frequent simultaneous interlocking contacts) demos. Collision responses in some specialized 3D scenes, such as those whose objects collide only in the 2D plane, can be computed using our approach. The 2D projection of each object onto the plane is used for the dynamics simulation. The left-image shows our method applied to a standard non-penetrating backtracking collision response method where contact points and normals are computed. All of the other simulations use the penalty-based collision response based on penetrating points and their penetration depths and directions. The right two images show collision responses between complex interlocking non-convex objects which are easily handled without specialized techniques such as convex decomposition.

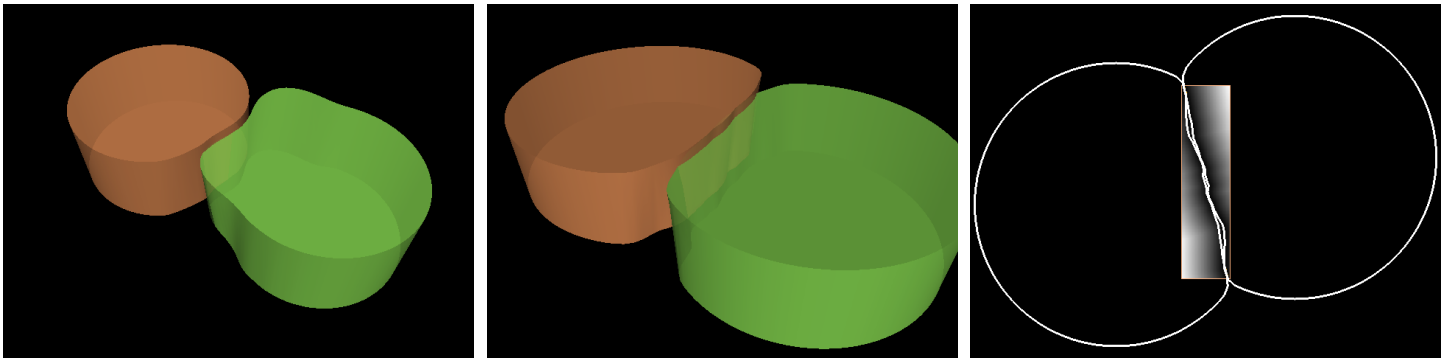

PLATE 3: **Wavy** (large deformable-bodies, continuous contact). Other important characteristics of our proximity query algorithm include not requiring any precomputation or complex data structures. Here we show proximity information being used for collision response between dynamically deformable bodies. The left image shows a wave propagating through the right object and hitting the left object. The collision response causes the object to become indented and creates a reaction wave in the left object. Many dynamics simulations resolve collisions by backing up the simulation to a moment before contact, we use a penalty-based method that applies a force at each penetrating point based on the amount of penetration. The center image shows the penalty-based response between two objects that were initially overlapping by a large amount. The right image shows the distance field around the contact area.