Ali Akgunduz*
Concordia University,
Department of Mechanical and Industrial Engineering,
1455 de Maisonneuve Blvd. West,
Montreal, Quebec H3G 1M8
phone (514) 848-2424 (Ext. 3179)
e-mail: akgunduz@me.concordia.ca

Prashant Banerjee
University of Illinois at Chicago,
Department of Mechanical and Industrial Engineering,
2039 ERF, 842 W. Taylor St.,
Chicago, IL 60607-7022
phone: (312) 996 5599
e-mail: banerjee@uic.edu

Sanjay Mehrotra
Northwestern University,
Industrial Engineering and Management Science,
2145 Sheridan Road, Tech C210,
Evanston, IL 60208
phone: (847) 491-3155
e-mail: mehrotra@iems.nwu.edu

A Linear Programming Solution for Exact Collision Detection

This paper addresses the issue of real-time collision detection between pairs of convex polyhedral objects undergoing fast rotational and translational motions. Accurate contact information between objects in virtual reality based simulations such as product design, assembly analysis, performance testing and ergonomic analysis of products are critical factors to explore when desired realism is to be achieved. For this purpose, fast, accurate and robust collision detection algorithms are required. The method described in the text models the exact collision detection problem between convex objects as a linear program. One of the strengths of the proposed methodology is its capability of addressing high speed interframe collision. In addition to the interframe collision detection, experimental data demonstrate that mathematical programming approaches offer promising results in terms of speed and robustness as well. [DOI: 10.1115/1.1846053]

Keywords: Interframe Collision Detection, Virtual Reality, Primal-Dual Interior-Point Method, Linear Programming

1 Introduction

Recent developments in both computer hardware and software have enabled virtual reality (VR) technology to become a powerful product development and analysis tool in engineering. VR technology has been utilized frequently in product testing, performance analysis, consumer research, and collaborative product development areas. Current trends indicate that in the near future the application area of VR will increase significantly. The major advantage of VR technology over traditional simulation techniques is its potential to simulate both users (such as a customer or a designer) and products in a common interactive environment. This virtual encounter is becoming much more powerful and realistic with the availability of advanced collision detection algorithms, data-gloves and haptic devices that are essential in order to perform basic VR functionalities such as touching, grabbing, feeling the surface of product, etc. Although extensive scientific work focusing on the problem of exact collision detection between virtual objects exists, the possibility of collision between fast moving objects has been mostly ignored. For instance, machining operations such as robots with high-speed cutters, the modeling of chemical reactions and air-flow for ventilation systems, and the use of lasers in medical surgeries involve lots of high-speed activities. Military equipment such as bullets, missiles, and rockets are also fast moving objects. Creating interactive animations of these objects in VR simulations necessitates the use of large traveling-steps between consecutive frames that frequently cause interframe collisions. Most existing collision detection methods fail to detect collisions between two consecutive frames. In this article, a real-time collision detection methodology between a pair of convex polyhedral objects undergoing fast rotational and translational motion will be presented. Areas of application for the proposed methodology can be categorized as (i) product development: auto body design, surface finish, design for assembly, quality control, performance testing, and ergonomic analysis and (ii) fast moving object design: high-speed impact tests, bullet-proof product development, anti-air weapon systems development, high-speed machining, molecular modeling, air/fluid modeling, computer games, movie industries, etc.

The paper is organized as follows. Section 2 summarizes earlier works in collision detection. The methodology is then described in Secs. 3 and 4 whereby Sec. 3 addresses problem formulation while Sec. 4 elaborates on the solution methodology. The fast, interframe collision problem is explained in Sec. 5. In Sec. 6, experimental results are presented. Lastly, concluding remarks are provided in Sec. 7.

2 Previous Work

Collision detection is a problem experienced in robotics, VR based product development and testing, interactive design, and other computer graphics applications. In computer graphics, emphasis is placed on determining algorithms that can detect collision in the presence of physical-based simulations, where motion is subject to dynamic constraints or external forces and cannot typically be expressed as a closed-form function of time [1–3]. In the context of computer graphics, a gain in the speed of solution for the collision problem directly translates into the size of scene that can be effectively managed in a software system where collision detection is necessary. It is important to note that for a complex scene, a very large number (hundreds of thousands to millions) of collision checks should be performed every second. Also, an ideal collision detection algorithm is the one that works universally well in a robust manner and under a wide variety of conditions—varying speeds of object movements between frames, situations with a large number of collisions because of high object densities in the scene or a low number of collisions in a sparsely populated scene.

*To whom correspondence should be addressed.

Contributed by the Engineering Simulation and Visualization Committee for publication in the JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING. Manuscript received May 21, 2004; Revised October 7, 2004. Associate Editor: N. Dorighi.

In order to develop efficient implementations for complex scenes, almost all of the collision detection approaches use a hierarchy of bounding volumes [2–4]. The bounding volumes, with simpler features, are used as a preliminary test for collisions. The absence of bounding volume collisions guarantees the absence of collision between corresponding objects and it is thus used to significantly reduce the number of features in the objects that are to be checked, especially if the objects are not close to each other. Choices of the bounding volume have often been the axis-aligned bounding boxes (AABBs) [5,6], the oriented bounding boxes [7], spheres [6,8], cones [4], s-bounds [9], and k-discrete orientation polytopes [10]. The reason for implementing simpler bounding volumes is that in these cases the collision computation can be performed in constant time [11]. In addition, it is often possible to efficiently transform these volumes as an object rotates and translates. While the use of bounding volumes is essential to speed up the computational times for collision checks when objects are far apart, the algorithms such as the Lin and Canny (LC) [12] closest feature algorithm or the Gilbert, Johnson, and Keerthi (GJK) [13] algorithm are needed when the objects are sufficiently close for the bounding volumes to declare collision.

The LC closest feature algorithm takes advantage of temporal coherence—the closest features (faces, edges, or vertices) change infrequently since the convex polyhedrons do not move swiftly along finely discretized paths. A candidate pair of features, one from each polyhedron, is determined based on Voronoi regions and whether closest points lie on these features. This is a local test involving only neighboring features of the current candidate features. If the test fails, the neighboring feature of one or both candidates is tried and the test is repeated. With some preprocessing, the algorithm can guarantee that every feature has a constant number of neighboring features. The GJK algorithm applies an iterative procedure to the Minkowski difference of the convex objects for which collision is being detected.

Applying the LC algorithm, Cohen et al. [11] utilized the temporal coherence assumption to speed up collision detection for complex scenes to develop a popular package known as I-COLLIDE. The underlying assumption is that the time steps are small enough so that objects do not travel large distances between frames. Using the temporal coherence assumption, they reduced the $O(n^2)$ possible interaction of n simultaneously moving objects to $O(n+m)$, where m is the number of objects for which the AABBs of objects overlap.

Multi-resolution modeling techniques have also been implemented for complex scene databases, such as model simplification, in order to compute bounding volume hierarchies [14]. The main idea in multi-resolution hierarchy is to compute and utilize a correspondence between the original model and the simplified one.

Recently, Ong and Gilbert [15,16] used the closest feature concept to develop more efficient implementations of the GJK algorithm. Regarding motion path planning with only two objects, they reported results similar to those obtained by the LC algorithm when motion paths did not collide and high temporal coherence existed. Also, a factor between two and ten times of improvement when temporal coherence is low was indicated. Mirtich [17] compared his Voronoi-clip (V-Clip) algorithm (based on LC approach) to the LC algorithm, Cameron's Enhanced GJK algorithm and his V-Clip algorithm for single pairs of objects. In Mirtich's [17] research, only floating point comparison was reported and no computational time comparisons are provided. Mirtich observes, however, that the edge-face and face-face collisions are two of the most problematic cases in the LC algorithm and that they account for the greater part of the coding complexity and cycling problems present. In the V-Clip algorithm, the edge-face case is simplified and the face-face problem is eliminated. Mirtich also emphasizes that there are robustness problems in the LC algorithm, the V-Clip algorithm and the Enhanced GJK algorithm. A fast and more robust version of GJK, Enhanced GJK, [18], was implemented. The fastest reported version known as SOLID (Software Library for Interference Detection) was developed [19,20]. Eventually, in 2000, Ehmann and Lin [21] introduced their new product, called SWIFT, which has proven to be the fastest algorithm the authors of this report have worked with so far. Although Luciano et al. [22] have indicated that SWIFT is unstable under certain conditions, it is still the fastest collision detection algorithm reported up to date.

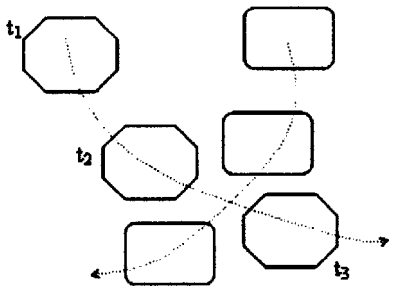


Fig. 1 Collision is missed due to large time steps

The state-of-the-art collision detection algorithms we reviewed in this paper ignore the possibility of interframe collision. Interframe collision is, however, a likely scenario to occur when fast-moving objects such as missiles, bullets, and spacecrafts are in the scene or when high-speed machining or air/fluid flow modeling are in consideration. Figure 1 illustrates an interframe collision in two-dimensional (2D) display. Xavier [23] modified the GJK algorithm to efficiently calculate the exact distances between objects when translational and rotational step-size were large. Xavier combines the vertices of an object in two consecutive time frames to generate a new convex object with twice the number of vertices from the original object. In his model problem size is increased by a factor of 2. Redon et al. [24] address fast continuous collisions. Their work is based on the integration of interval arithmetic and oriented bounding boxes. In their algorithm, first the collision is determined, then the exact time and location of the interframe collision is found. Determining the location and the time of exact collision is a backward simulation.

Similar to the Xavier's case, the mathematical programming approach introduced in this paper addresses high speed, interframe collision situations with minimal modification. We also introduce a number of heuristics to increase the efficiency of the interframe collision detection.

The possibility of using mathematical programming techniques for detecting collision in computer graphics has been known for some time. Although the works by Megiddo [25] and Seidel [26] do not address collision detection problems directly, their formulations helped researchers to correlate the linear programming and collision detection problem. However, in the existing literature these techniques are judged to be too inefficient for them to be useful for real-time computer graphics [7,27,28]. In the experiments conducted by the authors of this report, creative formulation of the collision problem are strengthened by efficient implementation using fundamental understanding of mathematical programming techniques and, consequently, promise to provide an alternative to solving collision detection problems. In terms of speed and robustness, the linear programming (LP) based approach for detecting collision among convex objects was found to perform as adequately as some of the popular techniques previously employed. These results are based on compact formulations of the problem coupled with a methodology that effectively implements an LP solver. Therefore, by taking advantage of the theoretical properties of the problem, solution times noticeably decrease.

3 Problem Formulation

In a virtual model, the objects are available in common formats such as *wrl*, *obj*, *dxf*, *iv*, *stl*, *flt* and others. In all of these

roughly equivalent formats, each object offers information about faces, vertices, vertex (or face) normals, materials, and textures. Hence, we assume that a list of vertices is given and the convex hull of the given vertices describes an object under consideration. In our modeling we use two virtual objects, or polytopes, called P_1 and P_2. For multiple objects, multiple pairwise collision detection tests are conducted.

The conceptual basis for the collision detection is to construct a convex hull using the vertices of the each object and to represent the collision as the intersection of convex hulls. Let P_1 be a polytop whose vertices are $\mathbf{v}_i$ where $i=\{1,\ldots,n_1\}$ and P_2 be a polytop whose vertices are $\mathbf{w}_j$ where $j=\{1,\ldots,n_2\}$ The terms polytope and object will be used interchangeably throughout the paper.

Detecting if polytopes P_1 and P_2 intersect is possible by determining if there exists a set of non-negative weights, $\boldsymbol{\alpha}$ for the vertices of P_1 and $\boldsymbol{\beta}$ for the vertices of P_2 that satisfies the following equations:

$$\sum_{j=1}^{n_1} \mathbf{v}_j\alpha_j - \sum_{j=1}^{n_2} \mathbf{w}_j\beta_j = 0 \tag{1}$$

$$\sum_{j=1}^{n_i} \alpha_j = \mathbf{1} \tag{2}$$

$$\sum_{j=1}^{n_i} \beta_j = \mathbf{1} \tag{3}$$

Equation (1) applies to each of the x, y, and z coordinates of the vertices for P_1 and P_2. These are represented as indicated here below:

$$\mathbf{v}_j=\begin{bmatrix} v_j^x \\ v_j^y \\ v_j^z \end{bmatrix}=\begin{bmatrix} x_j^1 \\ y_j^1 \\ z_j^1 \end{bmatrix}, \quad \mathbf{w}_j=\begin{bmatrix} w_j^x \\ w_j^y \\ w_j^z \end{bmatrix}=\begin{bmatrix} x_j^2 \\ y_j^2 \\ z_j^2 \end{bmatrix} \quad \text{and} \quad \mathbf{V}$$
$$=[\mathbf{v}_1 \ \cdots \ \mathbf{v}_{n1}], \quad \mathbf{W}=[\mathbf{w}_1 \ \cdots \ \mathbf{w}_{n2}]$$

Furthermore, the sum of weights for the combination of vertices of each object must equal to one, thereby resulting in Eqs. (2) and (3).

The values of the weights are unknown whereas the vertex coordinates are known. Since convex equations corresponding to both objects are considered simultaneously in Eqs. (1)–(3), the existence of a feasible solution indicates the presence of at least one common point on the two objects. This indicates a touch or an overlap signifying that a collision has occurred for the object scene corresponding to that particular time frame.

For noncollision situations, Eq. (1) is not satisfied. To sustain the feasibility, an artificial variable, φ, with coefficient $\mathbf{u}$ added to Eq. (1) where $\mathbf{u}$ represents the coordinates of distance between a point that is guaranteed to belong to P_1 and another point that is guaranteed to belong to P_2.

$$\mathbf{u}=\left(\frac{1}{n_2}\sum_{j=1}^{n_2} \mathbf{w}_j\right)-\left(\frac{1}{n_1}\sum_{j=1}^{n_1} \mathbf{v}_j\right) \tag{4}$$

Thus adding the Eq. (4) to Eq. (1), we obtain a convex hull equation, Eq. (5), which is always feasible (irrespective of collision or noncollision):

$$\sum_{j=1}^{n_1} \mathbf{v}_j\alpha_j - \sum_{j=1}^{n_2} \mathbf{w}_j\beta_j + \mathbf{u}\varphi = 0 \tag{5}$$

There is one $\mathbf{u}$ value for the each of the coordinates (x, y and z).

Equation (5) is always satisfied: When P_1 and P_2 overlap, then $\mathbf{u}\varphi=0$ and Eq. (5) becomes similar to the Eq. (1). When P_1 and P_2 are not overlapping, then $\mathbf{u}\varphi$ becomes positive to enable the following such that Eq. (5) is still satisfied:

$$\mathbf{u}\varphi = -\left(\sum_{j=1}^{n_1} \mathbf{v}_j\alpha_j - \sum_{j=1}^{n_2} \mathbf{w}_j\beta_j\right) \tag{6}$$

Finally, let

$$\mu = \max\{abs(\mathbf{u})\} \tag{7}$$

This is the coefficient in the objective function of the primal model.

In the following two sections we introduce the formulation of primal and dual models that are simultaneously solved by the interior-point algorithm will be introduced.

3.1 Primal Model. It is essential to establish a linear program whose solution can detect if P_1 and P_2 are colliding. This model is referred to as the primal model and is given as follows:

$$\min \mu\varphi \tag{8a}$$

s.t.

$$\sum_{j=1}^{n_1} \mathbf{v}_j\alpha_j - \sum_{j=1}^{n_2} \mathbf{w}_j\beta_j + \mathbf{u}\varphi = 0 \tag{8b}$$

$$\sum_{j=1}^{n_1} \alpha_j = 1 \tag{8c}$$

$$\sum_{j=1}^{n_2} \beta_j = 1 \tag{8d}$$

$$\varphi \geqslant 0; \quad \alpha_j \geqslant 0; \quad \beta_j \geqslant 0 \tag{8e}$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the primal decision variables.

In the above linear programming model we have five equations, three from each of the three coordinates of the vertices, and the remaining two ensuring that the weights in the convex hull representation sum to 1 for each object. The intersection of these two sets is nonempty when there is a collision; otherwise it is an empty set. Accordingly, when the two objects collide, the primal model has an optimal objective value 0, since in this case we have found a set of weights for P_1 and P_2 satisfying the Eq. (8a). Otherwise, the optimal objective value of the primal model is positive.

The methodology employed to solve the primal model is based on primal-dual interior-point algorithms for solving linear programs [29,30]. For accurate implementations of the interior-point algorithms, a positive feasible interior solution of the linear program being solved [Eq. (4)] is required. In this situation, $\mathbf{u}$ represents the vector joining the equally weighted "*center*" points of the two objects.

For simplicity let us denote the primal constraint matrix at a given time frame t by the following shorthand notation:

$$\mathbf{A}=\begin{bmatrix} v_1^x & \cdots & v_{n_1}^x & -w_1^x & \cdots & -w_{n_2}^x & u^x \\ v_1^y & \cdots & v_{n_1}^y & -w_1^y & \cdots & -w_{n_2}^y & u^y \\ v_1^z & \cdots & v_{n_1}^z & -w_1^z & \cdots & -w_{n_2}^z & u^z \\ 1 & \cdots & 1 & 0 & \cdots & 0 & 0 \\ 0 & \cdots & 0 & 1 & \cdots & 1 & 0 \end{bmatrix} \tag{9}$$

or $\mathbf{A}$ is written in block matrix form as shown here:

$$\mathbf{A}=\begin{bmatrix} \mathbf{V} & -\mathbf{W} & \mathbf{u} \\ \mathbf{I}_1 & \mathbf{I}_2 & \mathbf{z} \end{bmatrix}$$

where

$$\mathbf{I}_1=\begin{bmatrix} 1 & \cdots & 1 \\ 0 & \cdots & 0 \end{bmatrix}, \quad \mathbf{I}_2=\begin{bmatrix} 0 & \cdots & 0 \\ 1 & \cdots & 1 \end{bmatrix} \quad \text{and} \quad \mathbf{z}=\begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

For simplicity superscripts x, y, and z are removed from our notations for the remainder of the paper. Let us now denote the primal decision variable vector at a given time t as follows:

$$\boldsymbol{\gamma}^T=[\alpha_1 \ \cdots \ \alpha_{n_1} \ \beta_1 \ \cdots \ \beta_{n_2} \ \varphi] \tag{10}$$

If we replace the Eqs. (9) and (10) in the Eq. (8), the primal model at a given time t becomes as shown here:

$$\min \mu\varphi \tag{11a}$$

s.t.

$$\mathbf{A}\boldsymbol{\gamma}=\mathbf{b} \tag{11b}$$

$$\boldsymbol{\gamma}\geqslant 0 \tag{11c}$$

where $\mathbf{b}^T=[0\ 0\ 0\ 1\ 1]$ (T indicates the transpose)

At this point, when there is a collision, the objective function of the primal model is equal to zero (either $\mu=0$ or $\varphi=0$). However, when objects are away from each other, it is not possible to conclude the no-collision state from the primal model. To determine no-collision in a resourceful manner, the decision variables of the dual model are to be investigated.

3.2 Dual Model. We now define the dual constraint equations and objective function based on the primal constraint equations and objective function described earlier. There are five dual decision variables ($\boldsymbol{\pi}$) linked to five primal constraint equations. The dual objective function is to maximize the following:

$\sum_{i=1}^{5} b_i\pi_i$ which is the sum of last two dual variables.

Hence, the dual model is as follows:

$$\max \pi_4+\pi_5 \tag{12a}$$

s.t.

$$\mathbf{A}^T\boldsymbol{\pi}+\mathbf{s}=\mathbf{c} \tag{12b}$$

$$\mathbf{s}\geqslant 0 \tag{12c}$$

where $\boldsymbol{\pi}$ represents the decision variables for the dual model,

$$\boldsymbol{\pi}^T=[\pi_1 \ \ \pi_2 \ \ \pi_3 \ \ \pi_4 \ \ \pi_5].$$

$\mathbf{c}$ is a vector of $n+1$ elements of all zeros except the last element, which is μ, and $\mathbf{s}$ is a $n+1$ vector of slack variables.

In most situations, there are likely to be no collisions. In essence, the optimal objective value of the primal and the dual models will be positive. Under these circumstances, it is desirable to stop iterating early and not until an optimal solution is found. This is to be respected if $(\pi_4+\pi_5)>0$ is detected during an iteration of the primal-dual method. [In our formulation $(\pi_4+\pi_5)$ is the minimum possible distance between the tested polytopes.] Otherwise, the algorithm iterates until $\mu\varphi=0$. In practice, however, a small tolerance is to be used instead of zero to avoid numerical errors.

3.3 Initialization of Decision Variables. Primal decision variables are initially as presented in here:

$$\alpha_j^{t=0}=\frac{1}{n_1} \tag{13}$$

$$\beta_j^{t=0}=\frac{1}{n_2} \tag{14}$$

$$\varphi^{t=0}=1 \tag{15}$$

These equations ensure the feasibility of Eq. (8).

A dual feasible solution assists in the determination of initial dual variable values at the beginning. For the first solution of the collision detection problem, the first three dual variables are set to zero. The remaining of the two dual decision variables are determined from the constant η, which is the maximum absolute value among all x, y, z values of all the vertex coordinates from two objects.

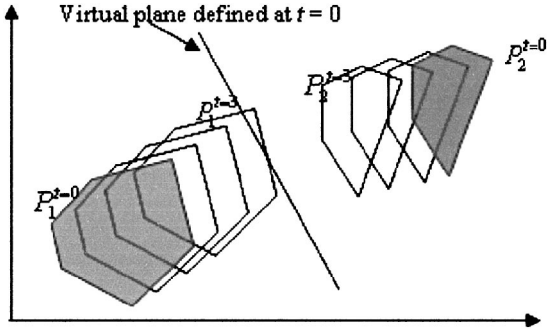


Fig. 2 Early collision determination using virtual plane

$$\pi_j^{t=0}=\begin{cases}0 & \text{if } j=\{1,2,3\}\\ \eta & \text{if } j=\{4,5\}\end{cases} \tag{16}$$

where t indicates the time. With the given initial values, it can be shown that the dual model is initially feasible.

4 Solution Methodology

A primal and dual model at each time frame is constructed by using the rotation matrix and translation vector of each object.

$$\mathbf{V}^t=\mathbf{R}_1\mathbf{V}^{t-1}+\mathbf{T}_1 \tag{17}$$

$$\mathbf{W}^t=\mathbf{R}_2\mathbf{W}^{t-1}+\mathbf{T}_2 \tag{18}$$

where t indicates time, $\mathbf{R}$ is the rotation matrix, and $\mathbf{T}$ is the translation vector. This model is then solved to determine collision at that frame. Sections 4.1–4.3 will discuss the procedure in order to obtain a solution.

4.1 Updating Decision Variables and Early Collision Detection. In the subsequent solutions, weights from previous frames are needed to initialize the primal decision variables as shown:

$$\boldsymbol{\alpha}^t=\boldsymbol{\alpha}^{t-1} \tag{19}$$

$$\boldsymbol{\beta}^t=\boldsymbol{\beta}^{t-1} \tag{20}$$

To update the dual decision variables in each frame, we use the following criteria:

$$\bar{\mathbf{s}}=\mathbf{c}-\mathbf{A}^T\boldsymbol{\pi}^{t-1} \tag{21}$$

$$\delta_s=\max\{-1.5\min\{\bar{s}_1,\bar{s}_2,\ \ldots,\bar{s}_n\},\varepsilon\} \tag{22}$$

$$\mathbf{s}=\bar{\mathbf{s}}+\delta_s \tag{23}$$

$$\pi_j^t=\begin{cases}\pi_j^{t-1} & \text{if } j=\{1,2,3\}\\ \pi_j^{t-1}-\delta_s & \text{if } j=\{4,5\}\end{cases} \tag{24}$$

where ε is a small positive number.

If any slack value is negative ($\min\{\bar{s}_1,\ldots,\bar{s}_n\}<0$), one must add a quantity of δ_s to every slack variable and subtract the same quantity from π_4 and π_5 to obtain consistency at the dual constraints.

If ($\delta_s<0$), the feasibility with the previous $\boldsymbol{\pi}$ values is not violated, so there is no need to check for collision in this frame. Feasibility guarantees that convex hulls do not overlap or have a common point; hence there is no collision.

Figure 2 illustrates the relationship between slack values and noncollision state in a 2D plane. As well, a virtual plane (separating plane) defined by $\boldsymbol{\pi}$ values at $t=0$ sustains the feasibility for three consecutive frames. At $t=3$, $\boldsymbol{\pi}^{t=0}$ no longer guarantees the no-collision state. Therefore, new $\boldsymbol{\pi}$ values should be calculated. This early determination of feasibility increases the efficiency of our algorithm. One must note that this check is not performed in the very first frame, however.

The first three elements of the dual decision variable $\boldsymbol{\pi}$ define a plane that separates two polytopes. Each vertex in the primal model maps to an equation of a plane in the dual space as follows:

i. the first n_1 constraints for $j=\{1,\ldots,n_1\}$ take the form of

$$v_j^x\pi_1+v_j^y\pi_2+v_j^x\pi_3+\pi_4+s_j=0 \tag{25}$$

ii. the next n_2 constraints for $i=\{1,\ldots,n_2\}$ take the form of

$$w_i^x\pi_1+w_i^y\pi_2+w_i^z\pi_3+\pi_5+s_{(n_1+i)}=0 \tag{26}$$

and when $\pi_4+s_j>0$ or $\pi_5+s_{(n_1+i)}>0$ holds for any vertex, then we conclude that the given vertex is the other side of the separating virtual plane and the noncollision cannot be guaranteed with the current dual decision variables. For more detailed information about the use of the separating plane concept in linear programming, one can refer to works by Megiddo [25,31] and Seidel [26].

4.2 Problem Size Reduction. In our model, the vertices with smaller slack values, $\bar{\mathbf{s}}$, indicate that these vertices are closer in terms of distance to the other object. This information is exploited in order to identify these vertices and to obtain a sublist for the problem. This enables one to work with much smaller problem sizes. The symbol ζ represents the new problem size. If the number of vertices is larger than 20, our algorithm reduces the problem size to 20 vertices.

4.3 Implementation of Primal-Dual Interior-Point Algorithm. Until a stopping criterion is satisfied, it is required to repeat the following interior point algorithm in five steps summarized here below:

1. Initialization of dual and primal variables (**X**, **S**, and **D**) where **X** and **S** are diagonal matrixes with diagonal elements $\boldsymbol{\gamma}$ and **s**, respectively. **D** is computed as indicated in Eq. (27):

$$\mathbf{D}=\mathbf{S}^{-1}\cdot\mathbf{X} \tag{27}$$

Since **X** and **S** are diagonal matrixes, neither the inverse of **S** nor matrix multiplication is computationally expensive.

2. Find the first derivative of the primal-dual affined scaling trajectory. This establishes a search direction for generating new trial points in one's quest for a solution. For the five elements of the dual scaling trajectory $\wp_\pi$, we use the outer product version of the Cholesky factorization as part of the 5×5 matrix inverse computations [32]. Next, an $(\xi+1)$ element column vector, $\wp_s$, of search direction values for the slack variables and $(\xi+1)$ element column vector $\wp_\gamma$ of search direction values for the primal decision variables are obtained as follows.

$$\wp_\pi=-(\mathbf{ADA}^T)^{-1}\mathbf{b} \tag{28}$$

$$\wp_s=-\mathbf{A}\wp_\pi \tag{29}$$

$$\wp_\gamma=\boldsymbol{\gamma}-\mathbf{D}\wp_s \tag{30}$$

3. Compute the primal and dual step factors (f_γ, f_s)

4. Create trial points. The new trial points are generated as follows.

$$\boldsymbol{\gamma}^{t+1}=\boldsymbol{\gamma}^t-f_\gamma\wp_\gamma \tag{31}$$

$$\mathbf{s}^{t+1}=\mathbf{s}^t-f_s\wp_s \tag{32}$$

$$\boldsymbol{\pi}^{t+1}=\boldsymbol{\pi}^t-f_s\wp_\pi \tag{33}$$

The robustness of the implementation is ensured by a suitable reduction in potential function (refer to Mehrotra [29] for more details). The central argument for potential function reduction is the fact that search directions are selected while maintaining primal and dual feasibility. The algorithm is observed to converge quickly—usually within one to three iterations when there is no collision and two to five iterations when there is a collision.

5. Check for the collision

If $(\mu\varphi\leqslant\boldsymbol{\varepsilon})$ $(\varepsilon=10^{-6}$ in our case)
Stop! There is a **COLLISION**
Else If $(\pi_4+\pi_5>0)$
Stop! There is **NO COLLISION**
Else
Go to step 1 and repeat the algorithm

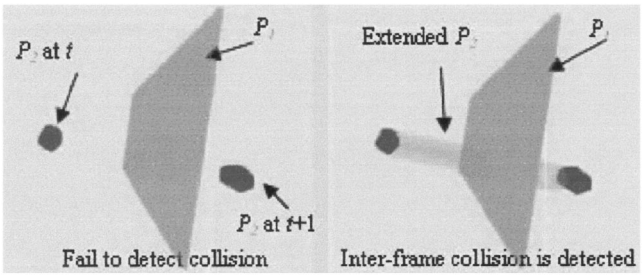


Fig. 3 Extended object assuming geometric coherence of frames

Since the interior point algorithm is implemented on selected vertices only, one must ensure that all unselected vertices are feasible with the obtained dual solution. For this purpose, Eq. (8) is tested for all the vertices to find the slack values $(\bar{\mathbf{s}})$. If any slack becomes negative, the vertex corresponding to the negative slack value is included into the selected vertex list, or subset. Vertices with larger positive slack values are removed from the subset as to maintain a reasonably small problem size. Finally, the interior point algorithm is applied to the new subset. To ensure the robustness of the algorithm under fast rotational and translational speeds, the check described here is performed:

Check all vertices of P_1 and P_2 for $j=\{1,2,\ldots,\xi\}$ as follows:

if $(\bar{\mathbf{s}}=\mathbf{c}-\mathbf{A}^T\boldsymbol{\pi}^t\leqslant0)$
Add the vertex to the subset, and remove the vertex with the largest positive slack value from the existing list.

If any vertex is added to the subset, repeat the algorithm with new sub-set and previous $\boldsymbol{\pi}$ values.

However, the previously performed experiments indicate that the addition of new vertices to the problem after a feasible solution is obtained is rarely the case.

5 Interframe Collision Detection

Most of the existing collision detection techniques have not addressed the problem of detecting collisions of very fast moving objects involving situations where the instance of collisions may not even be visible to the eyes when occurring between two frames. Redon et al. [24] use interval arithmetic and hierarchies of oriented bounding boxes to detect overlapping times between fast moving rigid objects. Their algorithm first checks collision between bounding boxes, then between the primitive, polyhedral, objects. In 1997, Xavier [23] reported that the GJK algorithm can actually be used in detecting collisions between fast moving objects. Parallel to Xavier's research, it was possible for the authors of this report to demonstrate that their collision detection algorithm can also address interframe collision problems efficiently. Due to this, a simple and novel approach was devised. It is based on the assumption that if two objects pass through the same region in a sufficiently small time frame, then we should consider that they have collided. Mathematically, this translates into linking the scene geometry of two successive time frames by taking the convex hull of vertex locations of an object at two successive frames. This is accomplished as described in the paragraphs to come.

The vertices of P_1 and P_2 are now to be represented by

$$\mathbf{V}^t \quad \text{and} \quad \tilde{\mathbf{W}}^t=[\mathbf{W}^{t-1}\ \ \mathbf{W}^t]$$

In the new form the total number of vertices is n_1+2n_2.

$$\mathbf{A}=\begin{bmatrix}\mathbf{V} & -\tilde{\mathbf{W}} & \mathbf{u}\\ \mathbf{I}_1 & \mathbf{I}_2 & \mathbf{z}\end{bmatrix} \tag{34}$$

The rest of the formulation is conceptually the same as before and is based on these modifications.

In Fig. 3, if P_1 (similarly P_2) is moving fast, then the vertices of P_2 are considered to be a collection of its vertices at two consecutive frames (current and previous frames) and an extended object is considered by taking the convex hull of these two vertex

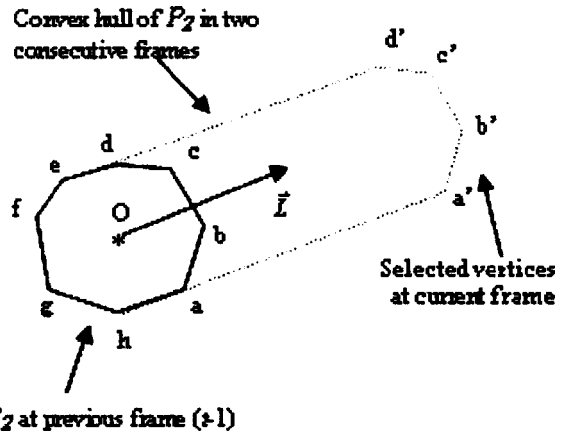


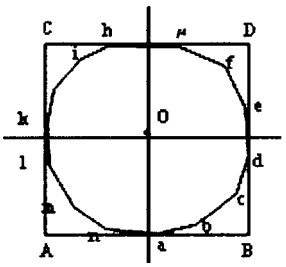
Fig. 4 Partially extended object assuming geometric coherence of frames. Only vertices {a, b, c, and d} are extended.

sets. Hence, the total number of vertices necessary to define extended P_2 is doubled from the original representation of P_2. One can construct a convex hull for P_2 by assuming a linear motion. Note that we are choosing one static and one fast moving object only to illustrate our approach better. When both objects are considered as fast moving, the described technique can still be applied to detect collision. In this case, one must define the convex hulls of both objects by the vertices from two consecutive frames.

5.1 Optimization. This section describes the hierarchy of optimizations that are added to the interframe collision detection system. Although the methodology introduced above can handle interframe collisions, doubling the problem size for fast moving objects is not desirable in practical applications due to the increased computational complexity. If we assume the rotational motion is minimal between consecutive frames while the translational speed is high, we can eliminate the need of doubling the problem size by adopting the following heuristic approach:

1. Check the collision between the extended bounding boxes. In this step, when there is a fast moving object in the scene, its bounding box (an oriented bounding box—OBB) is extended as described carlier and the first collision test is applied to the extended bounding boxes. If a collision is detected, we initiate the second step. Otherwise, we conclude that there is no collision. If one's goal is to find the exact collision time and location, first contact between the bounding boxes can be found by utilizing a similar simulation to the one introduced in Redon et al. [24].
2. When a collision is detected between the extended bounding boxes, there is a possibility of an interframe collision. Here we introduce an additional approach that can create a convex hull for the fast moving object in two consecutive frames without doubling the problem size. To achieve this goal we select the vertices with the closest distance proximities to the moving direction vector $\bar{L}$ (see Fig. 4 for a 2D illustration).

The selection of vertices {a, b, c, and d in Fig. 4} is mathematically straightforward. However, when the number of vertices belonging to the object increases, the operation becomes computationally expensive. This problem can be minimized by creating vertex groups during the initialization. A total of six groups are chosen for illustration purposes whereby one group is selected for each bounding box face. Figure 5 demonstrates the selection operation in 2D. If the face of the bounding box that is in closest proximity to the object's moving direction is determined, the object vertices grouped by the selected face are also in closest proximity to the direction vector, $\bar{L}$, as seen in Fig. 4. Once the subset of vertices is found, the convex hull for the extended object (P_2) is created with only the selected vertices.



2D BB Faces	Vertices grouped by the corresponding BB face
$\|AB\|$	l, m, n, a, b ,c ,d
$\|BD\|$	a, b, c, d, e, f, g
$\|DC\|$	e, f, g, h, i, j, k
$\|CA\|$	h, i, j, k, l, m, n

Fig. 5 Bounding box-vertex mapping for interframe collision detection

In a 3D environment, the determination of the closest face of the bounding box is performed as described here:

i. Find the distance proximity of each vertex from the center of bounding box. In Fig. 6, vertices B and D are the closest to the moving direction $\bar{L}$. Therefore the edge |BD| is chosen as the reference point in 2D to identify the vertex list to be extended.
ii. Within the 3D environment, a face is determined relative to the distance proximity of bounding-box edges.
iii. Once the face is determined, object vertices assigned to the face during initialization are chosen to extend the object. The matrix $\mathbf{Q} \in \mathbf{W}^t$ is next introduced to represent the selected vertices. If the total n_2' vertices are selected from the object, then the remaining $(n_2 - n_2')$ vertices will be defined with by the vector $\mathbf{Q}' \in \mathbf{W}^{t-1}$.

Now, this new form of the A matrix can be utilized to rewrite the LP model once more to solve the interframe collision problem represented as Eq. (35):

$$\mathbf{A} = \begin{bmatrix} \mathbf{V} & \mathbf{\Phi} & \mathbf{u} \\ \mathbf{I}_1 & \mathbf{I}_2 & \mathbf{z} \end{bmatrix} \tag{35}$$

where $\mathbf{\Phi} = [\mathbf{Q}\,\mathbf{Q}']$.

The rest of the formulation is conceptually the same, and is based on these modifications. In the new model, the problem size is not increased. Furthermore, the bounding-box and a list of vertices are grouped during initialization and are not recomputed in real-time during the collision check. Hence, the efficiency of the collision detection algorithm for fast moving objects is increased significantly. This extension for the interframe collision checking cannot, however, guarantee exact collision when the rotation of objects is extremely fast between two consecutive frames. We leave this issue to be addressed in a future work.

6 Experiments

Computational comparison of the results obtained from our approach, known as the Industrial Virtual Reality Institute Collision Detection Algorithm (IVRI-CD), with those obtained by running I-COLLIDE on a set of test problems is now considered. An OBB algorithm was developed and implemented for fair comparison with I-COLLIDE. For all the algorithms, the bounding-box option was turned ON. A library of 20 different objects ranging from 4 vertices (tetrahedron) to 264 vertices (sphere) was created for this experiment. Some of the objects were obtained from the set that was reported by I-COLLIDE, which had objects with up to 20

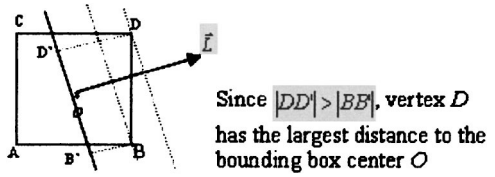


Fig. 6 Selection of bounding-box vertices closer to the direction of object movement

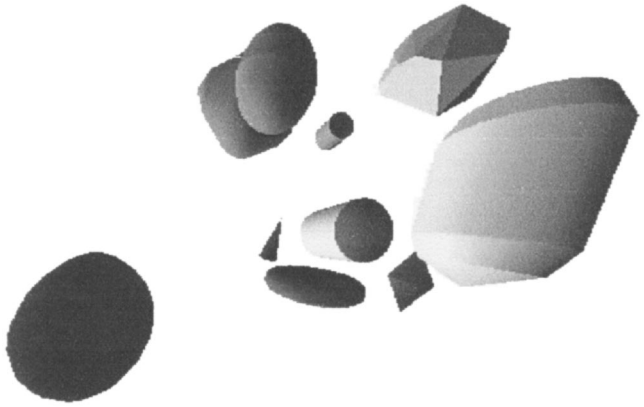


Fig. 7 Picture of the ten different randomly chosen objects

vertices, while the objects with higher vertex counts were generated by using ProEngineer by randomly choosing line segments and revolving them while ensuring the convexity. Some of these objects are shown in Fig. 7. Certain objects used in the experiments (e.g., sphere) are similar in spirit to the experimental set used by Mirtich [17], although the vertex counts are different. For our experiments, objects from these sets were randomly chosen—these details will be provided later on. The run times reported are in microseconds.

Figures 8–12 summarize our experimental results for comparing IVRI-CD and I-COLLIDE in various situations. In all the experiments, the objects were enclosed in a closed cubical space. Varying cubical space size was utilized for one of the experiments so that various densities of objects were generated. The tests were performed on a Silicon Graphics O2 workstation with IRIX 6.5 using C++ as a compiler tool. Details of these experiments are summed up in the following paragraphs.

Experiment 1: The first experiment involves pairwise comparisons. Two objects are initially placed at two opposite sides of the

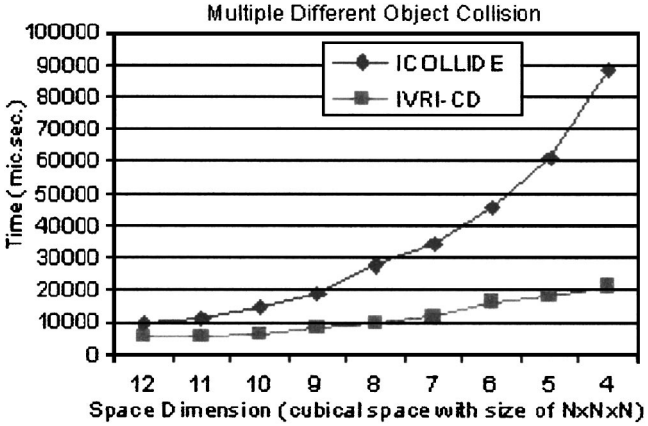


Fig. 10 Ten different objects under varying object densities determined by different sizes of the enclosing cubical space

cubical space. The motion involves translation and rotation along one of the faces of the cube. The objects are moved towards each other by 0.02 units per frame while rotating randomly about all three axes, based on a uniform distribution between 0 and 10 deg. The simulation for the each object pair is repeated 20 times. Figure 8 shows the relative performance of the two algorithms just before collision. An average of the last five frames before collision is used for timing. Since the bounding-box algorithms are able to detect noncollision while objects are apart from each other, the performance comparison of the two algorithms just before the collision is a lot more realistic. Since our algorithm uses the separating plane for early determination of the noncollision when objects are apart from each other, the performance of the IVRI-CD is much higher comparison to the performance of I-COLLIDE. When objects are subject to slow rotational and translational motion, the last five frames are roughly the points where bounding boxes of the observed objects are colliding. It is observed that,

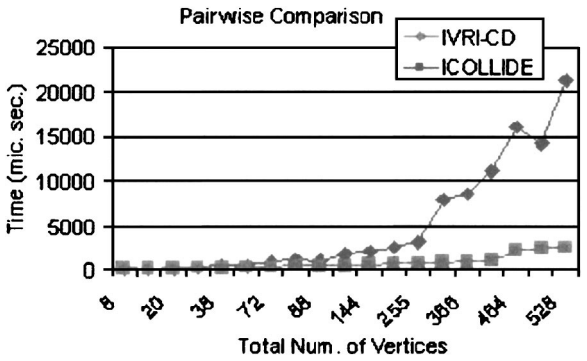


Fig. 8 Pairwise performance comparison just before the collision

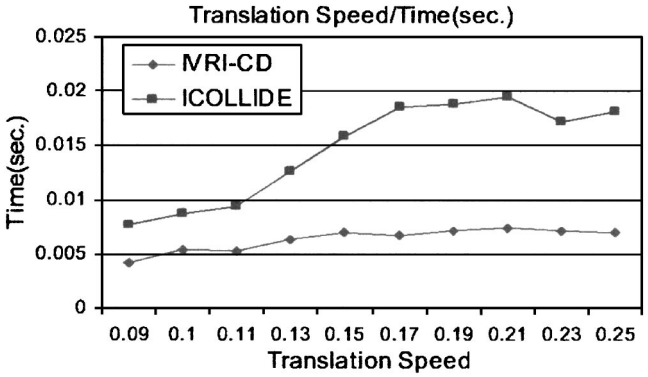


Fig. 11 Performance comparison when translational speed is increased

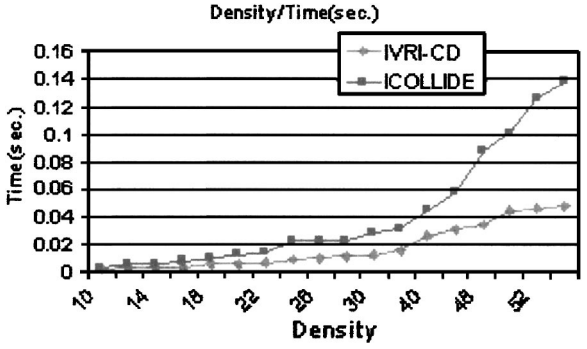


Fig. 9 Performance comparison when number of objects is increased

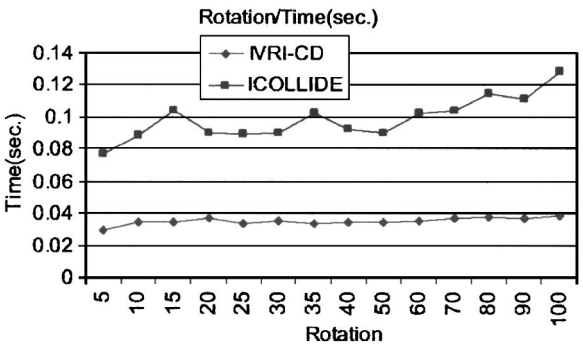


Fig. 12 Performance comparison when rotational speed is increased

despite the fact that the performance is roughly the same for objects with lower numbers of vertices; the IVRI-CD algorithm outperforms the I-COLLIDE algorithm significantly as the vertex count becomes large.

Experiment 2: For the second experiment, a number of identical objects (initially ten objects in the simulation) simulated in a cubical space where its length is 6 units. The experiment is repeated while the number of objects in the same cubical space is increased. The initial positions of objects in the cubical space are set randomly by a uniform distribution between 0 and the size of the cube for all three axes. Directions are determined randomly and the objects are free to penetrate each other. The rotations are determined by a uniform distribution between 0 and 10°. Simulation run length is kept at 1500 frames per density set. The final results are shown in Fig. 9. Here, it is observed that the IVRI-CD algorithm outperforms the I-COLLIDE algorithm when the number of objects in the scene increases.

Experiment 3: In the third experiment, the size of the cubical space is varied from $4\times4\times4$ to $12\times12\times12$ such that different densities of object packing within a scene are modeled. The size of the cube was changed in steps of 1. When the cube size was $4\times4\times4$, the objects were very densely packed and when the size was $12\times12\times12$, they were loosely packed. A total of ten different objects were randomly chosen from the library as illustrated in Fig. 7. For each cube size, statistics are collected for 1500 frames. The concluding results are given in Fig. 10 indicating that, as the density of the objects increases, the comparative performance of the IVRI-CD algorithm improves.

Experiment 4: Performance comparison when rotation and translation speeds are subject to change. Figures 11 and 12 summarize the performance of IVRI-CD under varying translational and rotational speeds.

7 Conclusions and Future Work

A linear programming approach for detecting collisions between convex objects that is robust and computationally promising has been developed. Although this algorithm uses well-known linear programming techniques, in the sense of collision detection it is formulized independently and is not dependent on any other ongoing collision detection research work. Within a short period of development time, the authors of this report were able to generate a model that clearly outperforms one of the most cited collision detection algorithms, I-COLLIDE. Recently, an approach for proximity queries based on distance computation was created [21] by combining the Voronoi-based tracking feature with a multi-level-of-detail representation to adapt to variations in levels of coherence and to speed up computations. The resulting software, entitled SWIFT, successfully exploits the closest features. Although the speed of detection collision is increased significantly, in comparison to I-Collide, it is reported that SWIFT has minor robustness problems. Luciano et al. [22] documented a number of occasions where SWIFT fails to detect collisions.

In conclusion, an alternative method of thinking for collision detection between convex objects has been initiated and it is equally competitive to some of the popular approaches in existence. Moreover, this model is naturally suited for detecting high-speed interframe collisions based on minimal changes to our problem formulation. Currently, the closest future algorithms are being exploited in our research to increase their efficiency so that they will run in near constant time.

Acknowledgments

The authors wish to thank the anonymous reviewers for their helpful comments on this paper. This work was supported in part by NSF grant No. 9988136 and NIST cooperative agreement 70NANB1H3014.

References

[1] Baraff, D., 1992, ''Dynamic Simulation of Non-Penetrating Rigid Bodies,'' Ph.D. thesis, Department of Computer Science, Cornell University, Ithaca, NY.
[2] Ponamgi, M., Manocha, D., and Lin, M. C., 1995, ''Incremental Algorithms for Collision Detection Between Solid Models,'' in *Proc. ACM/SIGGRAPH Symp. on Solid Modeling*, pp. 293–304.
[3] Ponamgi, M., Manocha, D., and Lin, M., 1997, ''Incremental Algorithms for Collision Detection Between Solid Models,'' IEEE Trans. Vis. Comput. Graph., **3**(1), pp. 51–64.
[4] Samet, S., 1989, *Spatial Data Structures: Quadtree, Octree and Other Hierarchical Methods*, Addison Wesley, Reading, MA.
[5] Hudson, T., Lin, M., Cohen, J., Gottschalk, S., and Manocha, D., 1997, ''V-COLLIDE: Accelerated Collision Detection for VRML,'' in *Proc. of VRML'97*.
[6] Quinlan, S., 1994, ''Efficient Distance Computation Between Non-Convex Objects,'' in *Proc. IEEE Conf. on Robotics and Automation*, pp. 3324–3329.
[7] Gottschalk, S., Lin, M., and Manocha, D., 1996, ''OBB-Tree: A Hierarchical Structure for Rapid Interference Detection,'' in *Proceedings of ACM SIGGRAPH*.
[8] Hubbard, P. M., 1995, ''Collision Detection for Interactive Graphics Applications,'' IEEE Trans. Vis. Comput. Graph., **1**(3), pp. 218–230.
[9] Cameron, S., 1991, ''Approximation Hierarchies and S-Bounds,'' in *Proc. Symposium on Solid Modeling Foundations and CAD/CAM Applications*, Austin, TX, pp. 129–137.
[10] Klosowski, J. T., Held, M., Mitchell, J. S. B., Sowizral, H., and Zikan, K., 1998, ''Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs,'' IEEE Trans. Vis. Comput. Graph., **4**(1), pp. 21–36.
[11] Cohen, J. D., Lin, M. C., Manocha, D., and Ponamgi, M., 1995, ''I-COLLIDE: An Interactive and Exact Collision Detection System for Large Scale Environments,'' in *Proc. ACM Int. 3D Graphics Conference*, pp. 189–196.
[12] Lin, M. C., and Canny, J. F., 1991, ''A Fast Algorithm for Incremental Distance Calculation,'' in *Proc IEEE Conf. on Robotics and Automation*, Vol. 2, pp. 1008–1014.
[13] Gilbert, E. G., Johnson, D. W., and Keerthi, S. S., 1988, ''A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space,'' IEEE Trans. Rob. Autom., **4**(2), pp. 193–203.
[14] Tan, T. S., Chong, K. F., and Low, K. L., 1999, ''Computing Bounding Volume Hierarchies Using Model Simplification,'' in *Proc. ACM Symposium On Interactive 3D Graphics*, pp. 63–70.
[15] Ong, C. J., and Gilbert, E. G., 1997, ''The Gilbert-Johnson-Keerthi Distance Algorithm: A Fast Version for Incremental Motions,'' in *Proc. IEEE Conf. on Robotics and Automation*, pp. 1183–1189.
[16] Ong, C. J., and Gilbert, E. G., 1999, ''The Gilbert-Johnson-Keerthi Distance Algorithm: A Fast Version for Incremental Motions,'' Technical Report, Univ. of Michigan, Ann Arbor.
[17] Mirtich, B., 1998, ''V-Clip: Fast and Robust Polyhedral Collision Detection,'' ACM Trans. Graphics, **17**(3), pp. 177–208.
[18] Cameron, S., 1997, ''Enhancing GJK: Computing Minimum and Penetration Distance Between Convex Polyhedra,'' in *Proc. IEEE Conf. on Robotics and Automation*, pp. 3112–3117.
[19] van den Bergen, G., 1999, ''A Fast and Robust GJK Implementation for Collision Detection of Convex Objects,'' J. Graphics Tools, **4**(2), pp. 7–25.
[20] van den Bergen, G., 1999, ''Efficient Collision Detection of Complex Deformable Models Using AABB Trees,'' J. Graphics Tools, **2**(4), pp. 1–14.
[21] Ehmann, S. A., and Lin, M. C., 2001, ''Accurate and Fast Proximity Queries Between Polyhedra Using Surface Decomposition,'' in *Eurographics. Computer Graphics Forum*, Vol. 20(3).
[22] Luciano, C., Banerjee, P., DeFanti, T., and Mehrotra, S., 2003, ''A Framework for Efficient and More Realistic Haptics Applications,'' in *Proceedings of 23rd ASME Computers and Information in Engineering Conference*, Chicago.
[23] Xavier, P. G., 1997, ''Fast Swept-Volume Distance for Robust Collision Detection,'' in *Proc. IEEE International Conference on Robotics and Automation*, Albuquerque, NM.
[24] Redon, S., Kheddar, A., and Coquillart, S., 2002, ''Fast Continues Collision Detection Between Rigid Bodies,'' Eurographics, Vol. 21(3).
[25] Megiddo, N., 1983, ''Linear-Time Algorithms for Linear Programming in R3 and Related Problems,'' SIAM J. Comput., **12**(4), pp. 759–776.
[26] Seidel, R., 1990, ''Linear Programming and Convex Hulls Made Easy,'' in *Proc. 6th Ann. ACM Conf. on Computational Geometry*, pp. 211–215, Berkeley, CA.
[27] Foley, J., van Dam, A., Feiner, S., and Hughes, J., 1993, *Computer Graphics*, 2nd ed., 5th printing, Addison Wesley, Reading, MA.
[28] Gallerini, R., and Sciomachen, A., 1993, ''On Using LP to Collision Detection Between a Manipulator Arm and Surrounding Obstacles,'' Eur. J. Oper. Res., **63**, pp. 343–350.
[29] Mehrotra, S., 1992, ''On the Implementation of a Primal-Dual Interior Point Method,'' SIAM J. Optim., **2**(4), pp. 575–601.
[30] Ye, Y., 1997, *Interior Point Algorithms: Theory and Analysis*, John Wiley, New York.
[31] Megiddo, N., 1984, ''Linear Programming in Linear Time When the Dimension is Fixed,'' J. Assoc. Comput. Mach., **31**(1), pp. 114–127.
[32] Golub, G. H., and van Loan, C. F., 1996, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore.