Parallel collision detection of ellipsoids with applications in large scale multibody dynamics

A. Pazouki*, H. Mazhar, D. Negrut

Department of Mechanical Engineering, University of Wisconsin-Madison, Madison, WI 53706-1572, USA
Emails: [pazouki, hmazhar, negrut]@wisc.edu

Abstract

This contribution describes a parallel approach for determining the collision state of a large collection of ellipsoids. Collision detection is required in granular dynamics simulation where it can combine with a differential variational inequality solver or discrete element method to approximate the time evolution of a collection of rigid bodies interacting through frictional contact. The approach proposed is structured on three levels. At the lowest level, the collision information associated with two colliding ellipsoids is obtained as the solution of a two-variable unconstrained optimization problem for which first and second order sensitivity information is derived analytically. Although this optimization approach suffices to resolve the collision problem between any two arbitrary ellipsoids, a less versatile but more efficient approach precedes it to gauge whether two ellipsoids are actually in contact and require the more costly optimization approach. This intermediate level draws on the analytical solution of a 3rd order polynomial obtained from the characteristic equation of two arbitrary ellipsoids. Finally, this intermediate level is invoked by the outer level only when a 3D spatial binning algorithm indicates that two ellipsoids share the same bin (box) and therefore could potentially collide. This multi-level approach is implemented in parallel and when executed on a ubiquitous Graphics Processing Unit (GPU) card scales linearly and yields a two orders of magnitude speedup over a similar algorithm executed on the Central Processing Unit (CPU). The GPU-based ellipsoid contact detection algorithm yields a 14 fold speedup over a CPU-based sphere contact detection algorithm implemented in the third party open source Bullet Physics Library (BPL). The proposed methodology provides the efficiency demanded by granular dynamics applications, which routinely handle scenarios with millions of collision events.

Keywords: Collision detection; Ellipsoid; Parallel computing

1. Introduction

Constraints in a multibody system can be classified as bilateral or unilateral. For holonomic systems, the former impose algebraic conditions that must be satisfied by the generalized coordinates used to capture the location and orientation of the rigid bodies in the dynamic system. They are associated with mechanical joints such as spherical, revolute, translational, etc., and are discussed extensively in [13]. Unilateral constraints are associated with non-penetration conditions that arise in frictional contact dynamics when using, for instance, a differential variational inequality (DVI) approach to numerically determine the time evolution of the multibody system [1]. Contact detection is critical for imposing the non-penetration condition, and its efficiency affects the overall performance of the multibody dynamics simulation, which at each integration time step requires the resolution of a contact detection problem.

Collision detection is critical when simulating the dynamics of granular material. Most of these simulations consider the grains as being either polyhedral or spherical. Polyhedral elements are useful for rocks/boulders while spherical elements are typically used for fine grain material such as sand. The main drawback of granular flow modeling with spherical bodies is that grain rolling can dominate the flow mechanism, resulting in low shear resistance. The variable-curvature shape of ellipsoids alters the rolling modes of bodies, which is conjectured to be responsible for differences between granular flow measurements and simulation results. By changing the three radii of ellipsoids independently, a wide range of microstructures can be obtained resulting in different macroscopic granular flow behavior. The use of ellipsoids provides an opportunity to investigate the anisotropic fabric of soil deposits [23].

* Corresponding author.

Additionally, the irregularity in the particle shape causes the mechanical interlocking of the particles, which tends to increase the fluidized bed voidages and minimum fluidization velocities [9, 19, 42]. In spite of the special features associated with the 3D granular flow of ellipsoidal bodies, spherical geometries have usually been preferred for the simulation of large numbers of 3D bodies due to the low computational burden incurred. The current contribution aims at addressing this issue by providing an efficient ellipsoid contact detection algorithm amenable to realistic simulations of large systems of granular material.

This paper is organized as follows. The next subsection provides a brief review of relevant literature. The subsequent three sections concentrate on the ellipsoid collision detection algorithm and its parallelization. Section 5 presents several numerical results that validate the algorithm and illustrate its scalability on the GPU. The last section summarizes findings and directions of future work.

1.1. Ellipsoid collision detection

Several types of ellipsoid geometry have been considered to date in collision detection tasks: classical ellipsoids [4-6], four-arc ellipsoid approximations [17, 20, 30, 36], and super-ellipsoids [4, 7, 38, 40]. In spite of their rather simple geometric representation, the salient attribute of the ellipsoid collision detection problem is its lack of an analytical solution. Typically, the numerical methods used rely on optimization algorithms or the solution of a series of systems of nonlinear equations. A common approach sets out by searching for the nearest pair of points $\mathbf{P}_1$ and $\mathbf{P}_2$ which satisfies the constraint of parallelism for the normal vectors at these points. The approach was originally posed as a six-variable constrained optimization problem [23] and subsequently extended to super-ellipsoids as an unconstrained optimization problem [38]. Improved performance can be achieved by approximating, when acceptable, the ellipsoids with simpler geometries. For instance, in the four arc ellipsoid approximation, the actual ellipsoid is replaced with a geometry obtained by revolving an ellipse approximated with four circular arcs [17, 20, 30, 36]. This approach results only in axisymmetric objects. In a similar fashion, ellipsoids were represented with clusters of spheres to simulate the flow of soybeans in [43]. Another ellipsoid contact detection approach draws on the mathematical concept of geometric potential whose minimization leads to the desired contact points [23, 33, 34]. In this method the geometric parallelism of the normal vectors at the contact point is only approximately fulfilled, and the approximation improves as the amount of penetration decreases. The penetration volume, contact area, and its shape are not calculated in this approach, which precludes its use in several classes of frictional contact dynamics problems that require these quantities. The geometric potential approach is implemented using a discrete function representation (DFR) in [16, 39]. In DFR, each body's surface is discretized by a number of points subsequently used to check for their inclusion in a potentially colliding body. Essentially, this method minimizes the geometric potential by direct search leading to an inefficient approach that is nonetheless attractive for contact detection of complicated surfaces. Finally, in [29] the contact normal is found by determining the intersection of an elliptic curve on the surface of one ellipsoid with the surface of the other ellipsoid and vice versa. Hence, the boundary of the region containing the contact point is determined. The location of the contact point is found by iterating this procedure. The resulting algorithm requires the solution of a system of nonlinear equations.

The most commonly used broad phase method for collision detection of ellipsoids is based on the contact of bounding spheres; i.e., spheres that completely enclose the space around an object [5]. Bounding boxes or rectangular swept sphere (RSS) volumes can also be used for the contact test [21]. RSS corresponds to a volume covered by a sphere whose center is swept across the surface of a 3D rectangle. As elaborated upon later, the proximity query used herein employs the algebraic criteria proposed specifically for ellipsoids [37]. These criteria are based on the sign of the roots of a fourth order polynomial whose coefficients are computed from the ellipsoids' radii, positions, and orientations. This method is implemented in [6] and has demonstrated good efficiency in continuous collision detection of ellipsoids.

Since the collision detection procedure pursued herein is intended to be used in large granular dynamics problems, approaches that leverage parallel computing are extremely desirable. This disqualifies several traditional methods, bringing to the forefront approaches which may be considered suboptimal in a sequential execution framework. One of the most widely used collision detection algorithms, discussed in [2] and [3], is a method known as "Sort and Sweep". This approach leverages the properties of Axis Aligned Bounding Boxes (AABBs) to detect collisions between objects of varying size. The basic premise of this algorithm is that if two objects are colliding, their projections/shadows will overlap on all three principal axes. Bounding boxes are generated for all objects and projected onto the global X, Y, and Z axes. Each axis is then sorted by position and traversed in linear time. A stack-like data structure is used to keep track of collision pairs; a collision will be recorded only if two bodies overlap on all three axes. This approach is simple to implement and is ideally suited for execution on the CPU. At most three parallel threads are used to sort and traverse the axes sequentially and only one thread can be used to compare the three lists to detect collisions, thus limiting its uses on parallel architectures. One salient feature of this algorithm is its ability to take advantage of temporal coherence. Since objects in a dynamics simulation rarely move large distances in one time step (the positions are temporally coherent), updates can be made to the sorted axes easily and with relatively little cost.

Parallel collision detection algorithms became popular with the onset of easily programmable GPUs. Many algorithms that use GPUs rely on shaders to do computations. Shaders are sets of instructions designed to perform calculations on polygons and textures, but they can also be used to perform basic computations. Several collision detection algorithms have been developed using shaders,

such as CULLIDE [12], R-CULLIDE [11], and Q-CULLIDE [10]. These algorithms could handle contacts between tens of thousands of polygons at near real-time speed and were faster than their CPU counterparts.

A parallel spatial subdivision algorithm for collision detection of spheres of equal size was introduced in [22]. It relies on the property that a sphere in a given bin in the spatially subdivided space can only interact with spheres in the 26 surrounding bins in a 3x3x3 grid. The bins are also sized larger than the spheres used. The algorithm presented in [22] was implemented on NVIDIA hardware using the CUDA Software Development Kit [27]. One unique feature, discussed at great length in [22] is the radix sort algorithm. Most collision detection algorithms require a fast sorting algorithm to arrange data into a usable form. Commonly used sorting methods such as quick sort, while fast on CPUs, are difficult to parallelize. Radix sort is not hindered by this drawback, as it is able to sort key-value pairs in parallel extremely efficiently. Additional work in [31] was done to improve the performance and scalability of the sorting algorithm, which was used in the GPU-based spatial subdivision method for collision detection proposed in [25]. The approach therein was designed to only determine sphere-to-sphere contacts and calculate the information needed by a Cone Complementarity Problem (CCP) [32] solver in a multibody dynamics simulation. The approach was successfully tested for contact detection of problems with more than 5 billion spheres and validated on smaller problems against the Bullet Physics Library (BPL) [8]. The ability to handle multi-billion body problems in [25] hinged on the use of GPU-computing. Since the ellipsoid collision detection algorithm proposed herein is designed for parallel execution on the GPU, this hardware platform and its relevance to scientific computing is briefly discussed next.

The GPU was originally designed for intense graphics processing, primarily encountered in video games, which require a dedicated hardware device capable of rendering, at high rates, hundreds of thousands of polygons at every frame. The computations performed are relatively simple and when enough fine grain parallelism can be exposed in an application, the GPU is capable of reaching extremely high FLoating-point OPeration (FLOP) rates. Specifically, the NVIDIA Tesla C2050 has a peak FLOP rate of 1.03 Teraflops in single precision and 0.515 Teraflops in double precision [28]. Comparatively, Intel core i7-965 CPU reaches 51.2 Gigaflops in single precision and 25.6 Gigaflops in double precision [18]. The GPU consists of several Streaming Multiprocessors (SMs). On an NVIDIA Tesla C1060 GPU, currently one of the most widely used GPU cards in scientific computing, each SM is made up of eight cores on which individual threads are processed in parallel. These cores are called Scalar Processors (SPs). Groups of two to three SMs are organized within a Texture Processing Cluster (TPC). The Tesla C1060 GPU contains 240 SP cores organized in 30 SMs grouped into 10 TPCs. This setup can manage simultaneously up to 30,720 parallel threads which the SPs swap between for execution with extremely low overhead due to hardware implemented support [24]. Threads on the GPU can use registers, which, like on the CPU, incur very little overhead. Generally, accessing a register is zero extra clock cycles per instruction, but delays may occur due to register read-after-write conflicts. On the GPU, each multiprocessor, along with having access to global/main memory (4 GB on Tesla C1060), has access to three other types of memory. The constant memory has extremely fast read times for cached data and is ideal for values that do not change during a kernel (function) call. Next, texture memory specializes in fetching (reading) of two dimensional textures; fetches are cached, increasing bandwidth if data is read from the same area. Lastly, shared memory is a smaller amount of extremely fast memory that is unique for each SM and shared by its SPs, unlike texture and constant memory that are global. This deep memory architecture was used to implement the ellipsoid collision detection algorithm described next.

2. Ellipsoid contact detection (the inner level)

The inner level of the proposed contact detection algorithm relies on the unconstrained optimization problem in [38]. For an efficient solution of this problem, the analytical expression of the Jacobian and Hessian are derived and later used in the minimization process. To this end, the positions (expressed in the global reference frame) of the contact points on two colliding ellipsoids are defined as explicit functions of an arbitrary unit vector. As a result, the parallelism constraint of the surface normal vectors at contact points is automatically satisfied. For example, for the given direction $\mathbf{c}$ in Figure , there is only one point on ellipsoid $\mathcal{E}_1$ whose surface normal vector has the same direction and orientation as $\mathbf{c}$ does. Similarly, there is a unique point on the surface of $\mathcal{E}_2$ associated with vector $\mathbf{-c}$.

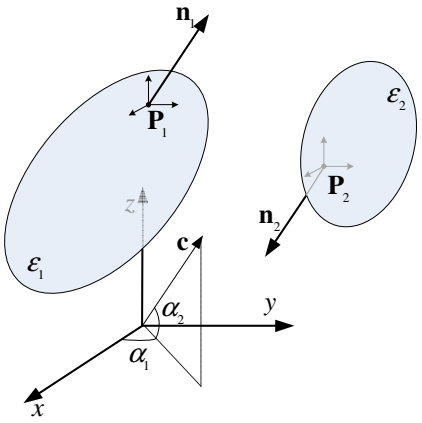


Figure 1 - Unit vector **c** and associated points on the ellipsoids. The orientation of ellipsoid i with respect to the global reference frame is described by an orientation matrix $\mathbf{A}_i$. A diagonal matrix $\mathbf{R}_i$ is defined based on the ellipsoid's radii: $\mathbf{R}_i = diag(r_{i1}, r_{i2}, r_{i3})$.

The vector $\mathbf{d}$ between $\mathbf{P}_1$ and $\mathbf{P}_2$,

$$\mathbf{d} = \mathbf{P}_1 - \mathbf{P}_2, \tag{1}$$

depends, together with $\mathbf{P}_1$ and $\mathbf{P}_2$, on the chosen direction defined by the unit vector $\mathbf{c}$, which in turn is a function of two parameterization angles (α_1, α_2). Therefore, an optimization problem is stated as

$$\min_{\alpha_1,\alpha_2} f(\alpha_1,\alpha_2) = \left\|\mathbf{d}(\alpha_1,\alpha_2)\right\|^2 = \mathbf{d}^T\mathbf{d} \equiv f(\mathbf{c}) \ . \tag{2}$$

Solving the minimization problem in Eq. (2) represents the inner level of the proposed methodology. It provides the contact points, contact direction, penetration depth, etc.; i.e., the information needed, for instance, in a dynamics simulation problem with frictional contact. By assuming the distance, f, as an explicit function of scalars α_1 and α_2, the associated first and second order sensitivities are obtained as

$$\frac{\partial f}{\partial \alpha_i} = 2(\mathbf{d}^T \frac{\partial \mathbf{d}}{\partial \alpha_i}), \tag{3}$$

$$\frac{\partial^2 f}{\partial \alpha_i \partial \alpha_j} = 2[(\frac{\partial \mathbf{d}}{\partial \alpha_i})^T(\frac{\partial \mathbf{d}}{\partial \alpha_j}) + \mathbf{d}^T(\frac{\partial^2 \mathbf{d}}{\partial \alpha_i \partial \alpha_j})], \tag{4}$$

where $i, j \in \{1,2\}$ and $\mathbf{c} \equiv \mathbf{c}(\alpha_1,\alpha_2) = (\cos\alpha_1 \cos\alpha_2, \sin\alpha_1 \cos\alpha_2, \sin\alpha_2)^T$ is a unit vector defined based on α_1 and α_2. The essential ingredients to evaluate the sensitivities in Eqs. (3) and (4) are derived in the Appendix (see Eqs. (A-10) and (A-11)). The first order partial derivatives in Eq. (3) are essential for all gradient-based optimization methods. The second partial derivatives of Eq. (4) are useful for some more advanced optimization techniques [41]. The outcome of the optimization problem is the pair of points $\mathbf{P}_1$ and $\mathbf{P}_2$ and normal vectors $\mathbf{n}_1$ and $\mathbf{n}_2$, which together are used to determine the contact state of the ellipsoids. The final stage of the ellipsoid contact detection algorithm is shown in Figure [38]. If the ellipsoids overlap or touch each other, the method described above produces the depth of penetration $\|\mathbf{d}\|$ and the contact condition becomes

$$\mathbf{d}^T\mathbf{n}_1 \geq 0. \tag{5}$$

In this case, the midpoint of the segment connecting $\mathbf{P}_1$ and $\mathbf{P}_2$ is chosen as the contact point that is returned to the dynamics simulation engine along with $\mathbf{n}_1$ and $\mathbf{n}_2$. However, if one ellipsoid is completely contained within another, the center of the former is chosen as the contact point. This step is essential for the spatial subdivision method in which the contact point should belong to both bodies. Similarly, the separation condition of a pair of ellipsoids is stated as

$$\mathbf{d}^T\mathbf{n}_1 < 0. \tag{6}$$

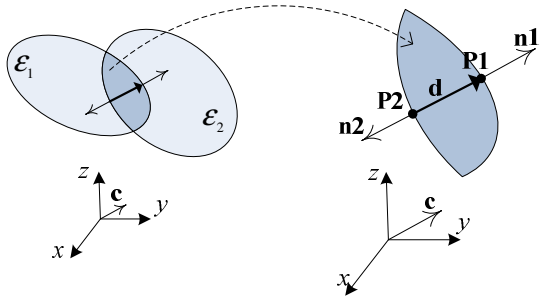


Figure 2 - Two ellipsoids in contact, nomenclature

3. Determining the contact state of two ellipsoids (the intermediate level)

At the inner level of the proposed methodology, the optimization-based contact detection approach outlined in the previous section and called herein the "Optimization-based Method (OpM)", is computationally expensive. However, it provides all the information needed in a dynamics simulation engine to characterize the frictional contact forces associated with the contact of the two ellipsoids. Note that OpM can be used to gauge the contact state of any two ellipsoids. As illustrated shortly in the numerical experiments section, using OpM in this fashion leads to a suboptimal algorithm. A state of contact between two arbitrary ellipsoids is a relatively rare event, and this works against OpM since in reaching the no-contact conclusion costly information is generated ($\mathbf{P}_1$ and $\mathbf{P}_2$ and normal vectors $\mathbf{n}_1$ and $\mathbf{n}_2$) only to be immediately discarded. A second, more expeditious, test is introduced at an intermediate level to understand whether two ellipsoids are in contact or not. The approach referenced herein as the "Characteristic equation-based Method (ChM)" implements this test. Specifically, it draws on the characteristic equation associated with a pair of ellipsoids to expeditiously determine the state of contact; i.e., to produce a "yes/no" answer to the question "Are two arbitrary ellipsoids in contact?". When a state of contact is positively identified, OpM is employed to compute the attributes of the contact.

The representation of an ellipsoid $\mathcal{E}$ in the global reference frame is given by $\mathcal{E}: \mathbf{Y}^T\mathbf{Q}\mathbf{Y} = 0$ where $\mathbf{Y} = (X, Y, Z, 1)^T$,

$$\mathbf{Q} = \begin{pmatrix} \mathbf{A}(\mathbf{R}^{-1})^2\mathbf{A}^T & -\mathbf{A}(\mathbf{R}^{-1})^2\mathbf{A}^T\mathbf{b} \\ -\mathbf{b}^T\mathbf{A}(\mathbf{R}^{-1})^2\mathbf{A}^T & \mathbf{b}^T\mathbf{A}(\mathbf{R}^{-1})^2\mathbf{A}^T\mathbf{b} - \mathbf{1} \end{pmatrix}, \tag{7}$$

and $\mathbf{b}$ is the translation of the ellipsoid local, principal, and centroidal reference frame in the global reference frame (see also Eqs. (A-1) and (A-2)). The characteristic equation of a pair of ellipsoids is defined as

$$\bar{g}(\xi) = \det(\xi\mathbf{Q}_1 + \mathbf{Q}_2). \tag{8}$$

After a rescaling by the coefficient of the highest degree term, the characteristic equation becomes

$$g(\xi) = \xi^4 + b_3\xi^3 + b_2\xi^2 + b_1\xi + b_0. \tag{9}$$

Several relevant remarks in relation to the roots of the characteristic equations are stated/proved in [37]:

(i) The characteristic equation has at least two negative roots.

(ii) The two ellipsoids are separated by a plane if and only if the characteristic equation has two distinct positive roots.
(iii) The two ellipsoids touch each other externally if and only if the characteristic equation has a positive double root.

Starting with these remarks, a new set of conditions based on the roots of the derivative of the characteristic equation was developed to implement ChM. The derivative of the characteristic equation is a third degree polynomial, therefore much simpler to solve analytically. The roots of the derivative of the characteristic equation can be found by solving

$$\zeta^3 + a_2\zeta^2 + a_1\zeta + a_0 = 0, \tag{10}$$

The following rules can be used to implement ChM; i.e., to completely answer the question "Are two arbitrary ellipsoids in contact?":
(a) The necessary conditions for g to have two positive roots are:

(1) $b_0 = \det(\mathbf{Q}_2) > 0$.

(2) $\chi = -18a_2a_1a_0 + 4{a_2}^3a_0 - {a_2}^2{a_1}^2 + 4{a_1}^3 + 27{a_0}^2 < 0$

This condition implies that g' has three distinct roots. Assume the roots as $\zeta_1 < \zeta_2 < \zeta_3$ (where $\zeta_1 < 0$).

(3) $\zeta_3 > 0$

(b) If any of the conditions listed in (a) is violated, the ellipsoids are in contact. On the other hand, if all of them are satisfied, the following two criteria will be sufficient to determine the sign of the roots of g:

(1) If $\zeta_2 > 0$, $g(\zeta_2)g(\zeta_3) < 0$ is a sufficient condition for g to have two positive roots; $g(\zeta_2)g(\zeta_3) > 0$, is a sufficient condition for g not to have two positive roots. If $g(\zeta) = 0$ at $\zeta > 0$, g has a positive double root located at $\zeta > 0$.

(2) If $\zeta_2 < 0$, although g has at least one positive root, further calculation is required to obtain the sign of the only remaining root (it already had two negative roots). The remaining root can be searched in the interval $[\zeta_2, \zeta_3]$. Since there is one and only one root in this interval, it can be found by a simple root finding method like bisection. Moreover, it is not essential to find the exact value of the root: the bisection algorithm can be terminated when the interval bounds have the same sign.

These criteria are sufficient to determine the state of intersection of two ellipsoids; i.e., to implement ChM.

4. Parallel contact detection

The parallel contact detection approach is implemented on the GPU and draws on a *spatial subdivision* idea [25] modified to accommodate the specifics of the problem at hand. The strategy is to divide the space into bins and perform an exhaustive collision detection per bin that checks all pairs of bodies that intersect the considered bin. This exposes a level of parallelism that is called herein the "outer level", and which can be controlled through the bin size. An optimal choice of the bin size is a topic discussed in [25]. Two methods were implemented on both the CPU and GPU: Method-1 uses OpM exclusively; Method-2 uses the combination OpM and ChM. The spatial subdivision approach consists of ten stages, which are employed for both Method-1 and Method-2.

Stage 1: The purpose of this stage is to find the number of bins intersected by each body in the problem. Each body is processed in parallel with a single thread. An array of size equal to the number of bodies is considered. The total number of bins intersected by each body is stored in the body-associated element of the array.

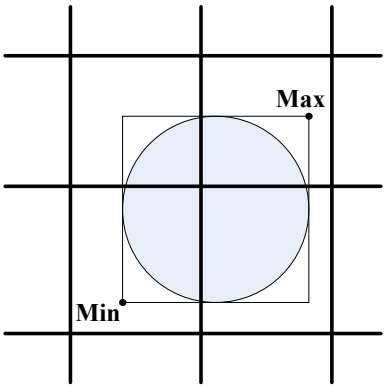


Figure 3 - Bin-body intersection and location of Min and Max points

In the case of spheres represented in a global reference frame, the minimum and maximum points of the bounding box of each sphere are calculated by subtracting or adding the radius of the sphere to its center coordinates respectively (Figure), allowing for fast resolution of the body-bin intersection question [25]. For an ellipsoid, identifying the minimum and maximum points for the bounding box draws on the parameters defined in the optimization-based approach. Specifically, the x-coordinate of the maximum point can be obtained by considering the plane tangent to the ellipsoid and parallel to the Y-Z plane. Therefore, it is the same as the x-coordinate of the maximum point $\mathbf{P}$ calculated from Eq. (A-13) by considering $\mathbf{c} = (1,0,0)^T$. For the y and z coordinates, $\mathbf{c}$ should be $(0,1,0)^T$ and $(0,0,1)^T$, respectively. For the minimum point, $\mathbf{c}$ is the negative of the vectors mentioned above. Therefore, considering the components of matrix $\mathbf{M}$ of Eq. (A-11) as

$$\mathbf{M} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}, \tag{11}$$

the Min and Max points are expressed as

$$\begin{matrix} \text{Max} \\ \text{Min} \end{matrix} = \pm \left(\sqrt{m_{11}}, \sqrt{m_{22}}, \sqrt{m_{33}}\right)^T + \mathbf{b}, \tag{12}$$

Stage 2: A parallel inclusive scan operation is carried out on the array of Stage 1 to find out the total number s_1 of bin-body intersections. This stage allocates memory on the GPU for an array of key-value pairs of length s_1. The parallel GPU scan operation used is the one provided by the *Thrust* library [15].

Stage 3: Similar to stage 1, but instead of counting the bin-body intersections, the bin-body intersections are now saved into the key-value array of size s_1. The value is the body index and the key is the index of the intersected bin.

Stage 4: Using radix sort, the array of pairs of stage 3 is sorted in parallel based on the key value. In what follows, this array is denoted by $\mathbf{B}$. The parallel sorting utility used is provided by the *Thrust* library [15]. In array $\mathbf{B}$ the keys hold duplicates of bin indices as each bin may be intersected by multiple bodies. These duplicates coalesce upon the completion of the sort.

Stage 5: At this stage a parallel reduction is performed on $\mathbf{B}$ to reduce the sequence of similar bins and find the total number s_2 of active bins. The size of each sequence of similar indices is stored as the value in another key-value array, $\mathbf{C}$. The parallel reduction utility used is provided by the *Thrust* library [15].

Stage 6: A parallel inclusive scan operation is carried out on the value entries of array $\mathbf{C}$. The outcome of this stage is written to the key component of array $\mathbf{C}$ and will be used in conjunction with $\mathbf{B}$ to access the list of bodies intersected by each active bin.

Stage 7: Using radix sort, array $\mathbf{C}$ is sorted in parallel based on the value. Therefore, the bins with the minimum difference in the number of contacts, group together. The purpose of this stage is to improve the load balancing in the GPU based contact detection.

Stage 8: At this stage, an array $\mathbf{E}$ of size s_2 is allocated to store the number of actual contacts in each active bin. Knowing the set of ellipsoids touching a particular bin from arrays $\mathbf{B}$ and $\mathbf{C}$, the total number of actual contacts per active bin is determined and stored into the associated component of $\mathbf{E}$. To this end, each parallel thread processes one bin and performs brute force contact detection for the ellipsoids that intersect this bin using OpM. If the ellipsoids are in contact and the contact point is inside the bin, that contact is counted and reflected in the array $\mathbf{E}$.

Stage 9: A parallel inclusive scan operation is done for $\mathbf{E}$ to find out the total number s_3 of contacts.

Stage 10: An array $\mathbf{F}$ of size s_3 is allocated to hold the actual ellipsoid collision information data (location of contact point, normal unit vector at point of contact, etc.). The remaining work of this stage is similar to Stage 8. However, instead of only counting the contacts, the actual contact data is computed and stored in $\mathbf{F}$.

This sequence of ten stages reflects the implementation of Method-1. Register and shared memory GPU hardware constraints along with a larger memory footprint required by Method-2 result for the latter in three additional sub-stages executed right after Stage 7:

Stage Post 7-a: An array of size s_2 is considered and ChM is invoked to obtain the number of contacts per active bin.

Stage Post 7-b: Parallel scan is performed to obtain the total number s_4 of contacts.

Stage Post 7-c: Array $\mathbf{D}$ of size s_4 is allocated. The remaining work is similar to Stage 7-a but instead of counting the contacts, the indices of the contacting pairs are stored in $\mathbf{D}$. Note that in this case, instead of testing all possible pairs, stages 8 and 10 draw on OpM to investigate a small number of potential contact scenarios stored in $\mathbf{D}$.

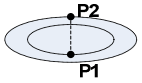
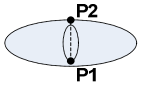
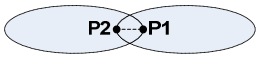
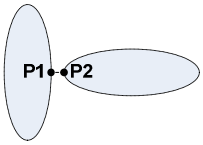
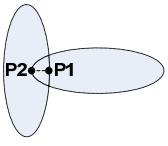
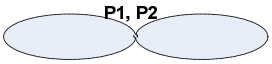
5. Algorithm validation: comparison to state of the art

The following subsections briefly describe the validation process for ChM, OpM, and the overall algorithm, which is validated against BPL [8]. BPL is also used to gauge the performance of the parallel CUDA-based [27] ellipsoid collision detection implementation.

5.1. Validation of OpM and handling of special cases

OpM was implemented using a gradient descent method with optimal step size [41]. Its validation was carried out using a collection of cases for which an analytical solution of the contact problem could be formulated. Of these cases, six cases with special relative poses are described and illustrated in Table . The contact data obtained by OpM is reported in the "Calculated Contact Data" column and indicates how OpM handles some limit cases such as complete and exact overlap, ellipsoid completely contained inside another ellipsoid, etc.

Table 1 - Parameters used to verify the state of contact of two ellipsoids: Optimization approach

	Body 1	Body 2	Calculated Contact Data	Schematic Configuration
Case 1	$(r_{11}, r_{12}, r_{13}) = (3,1,2)$ Rotation: 0° $\mathbf{b}_1 = (1,1,1)$	$(r_{21}, r_{22}, r_{23}) = (6,2,4)$ Rotation: 0° $\mathbf{b}_2 = (1,1,1)$	$\|\mathbf{d}\| = 3$ $\mathbf{c}^T = (0,1,0)$	
Case 2	$(r_{11}, r_{12}, r_{13}) = (3,1,2)$ Rotation: 90° about **z** $\mathbf{b}_1 = (1,1,1)$	$(r_{21}, r_{22}, r_{23}) = (12,4,8)$ Rotation: 0° $\mathbf{b}_2 = (1,1,1)$	$\|\mathbf{d}\| = 7$ $\mathbf{c}^T = (0,1,0)$	
Case 3	$(r_{11}, r_{12}, r_{13}) = (3,1,2)$ Rotation: 0° $\mathbf{b}_1 = (0,0,0)$	$(r_{21}, r_{22}, r_{23}) = (3,1,2)$ Rotation: 0° $\mathbf{b}_2 = (5.8,0,0)$	$\|\mathbf{d}\| = 0.2$ $\mathbf{c}^T = (1,0,0)$	
Case 4	$(r_{11}, r_{12}, r_{13}) = (3,1,2)$ Rotation: 90° about **z** $\mathbf{b}_1 = (1,1,1)$	$(r_{21}, r_{22}, r_{23}) = (1,3,2)$ Rotation: 90° about **z** $\mathbf{b}_2 = (5.2,1,1)$	$\|\mathbf{d}\| = 0.2$ $\mathbf{c}^T = (1,0,0)$	
Case 5	$(r_{11}, r_{12}, r_{13}) = (3,1,2)$ Rotation: 90° about **z** $\mathbf{b}_1 = (1,1,1)$	$(r_{21}, r_{22}, r_{23}) = (1,3,2)$ Rotation: 90° about **z** $\mathbf{b}_2 = (4.8,1,1)$	$\|\mathbf{d}\| = 0.2$ $\mathbf{c}^T = (1,0,0)$	
Case 6	$(r_{11}, r_{12}, r_{13}) = (3,1,2)$ Rotation: 0° $\mathbf{b}_1 = (0,0,0)$	$(r_{21}, r_{22}, r_{23}) = (3,1,2)$ Rotation: 0° $\mathbf{b}_2 = (6,0,0)$	$\|\mathbf{d}\| = 0$ $\mathbf{c}^T = (1,0,0)$	

5.2. Validation, parallel ellipsoid collision detection

Unlike spherical collision detection where an analytical solution to the problem is readily available, the ellipsoid collision detection validation is slightly more problematic due to its lack of an analytical solution. Since no utility was found to assist with the numerical validation of the overall ellipsoid collision detection algorithm, the numerical experiments were carried out assuming that each ellipsoid i had three identical radii, $r_{i1} = r_{i2} = r_{i3}$. Although the geometries became spheres of various radii, they were still treated using Method-1 or Method-2. The results could then be validated against two codes for spherical contact detection, the open source BPL and the code described in [25]. While BPL provides a good reference for sphere-sphere contact detection, it does not provide support for the ellipsoid geometry. Specifically, the ellipsoid contact detection process falls back onto the case of handling a general convex shape where the geometry is represented by a set of boundary nodes on a pre-defined faceted mesh.

The number of contacts obtained from OpM, ChM, BPL, and the implementation in [25] was identical, as reported in Table . Moreover, the error in the contact data, such as contact point position and contact normal vector, was low. For example, for the test case with 262,144 bodies, the distribution of the relative error in contact point location, defined as

$$Relative\ Error = \frac{contact\ \ point\ \ location\ \ error}{min(r_i, r_j)} \times 100, \tag{13}$$

is shown in Figure , where r_i and r_j are radii of the contacting spheres. All results reported in this paper are based on 16 iterations in the optimization algorithm. The quality of the results can be improved further by tuning the optimization algorithm parameters such as number of iterations.

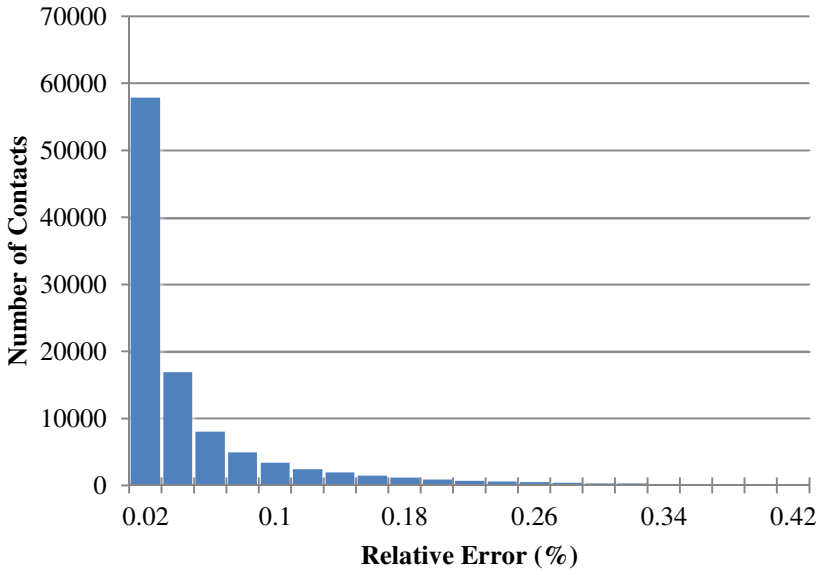


Figure 4 - Example relative error distribution in contact detection of 262 144 bodies with 105 920 contacts

Table reports the number of contacts for different numbers of spheres. Sphere centers were generated randomly within a cube which was sized such that the number of spheres per unit volume was $n = 1.25 \times 10^{-7}$ for all tests. Radii were randomly chosen in the range $[0,101]$, where all units are SI. Note that the goal was only to validate the number of collisions and there was no interest in determining collision information.

Table 2 - The number of contacts obtained with the four methods for different number of bodies

	Number of contacts		
Spheres	Ellipsoid Algorithm	Bullet	Sphere

	OpM	ChM	Physics Library [8]	Algorithm [25]
128	37			
1 024	335			
8 192	2 969			
32 768	11 670			
131 072	49 000			
262 144	105 920			
524 288	190 747			
1 048 576	487 246			
1 677 696	855 770			
2 097 152	1 064 984			

5.3. Parallel collision detection performance

The proposed GPU parallel ellipsoid contact detection methodology was compared to a sequential version executed on the CPU, which used the same spatial subdivision approach. The GPU implementation used CUDA [26] and was exercised on an NVIDIA TESLA C2070 with 6GB of memory and 448 scalar processors. The CPU algorithm was tested on an Intel Core 2 Duo E8500. The GPU and CPU tests were run for the same data sets, as described earlier; moreover, both were single precision implementations. As illustrated in Figure and Figure , the CPU scaling of Method-1 and Method-2 with the number of contacts is linear.

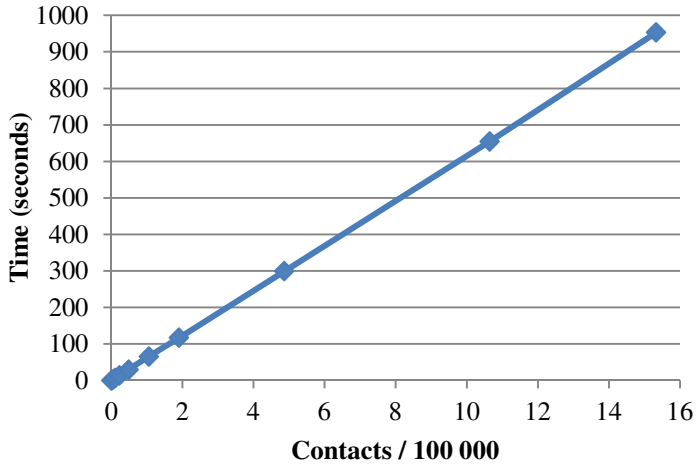


Figure 5 - Relationship between number of contacts and collision time for CPU based implementation of Method-1

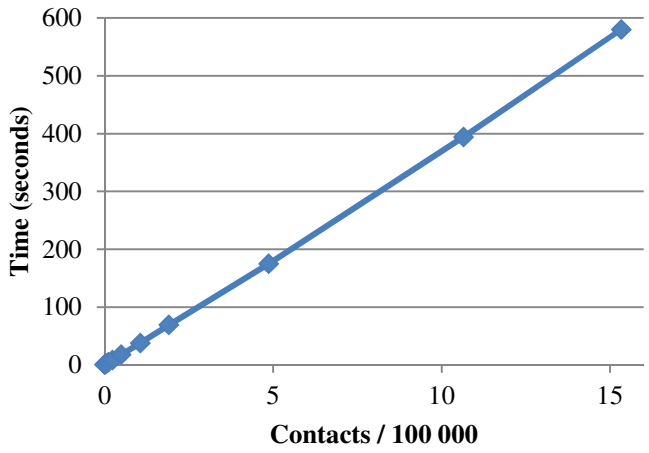


Figure 6 - Relationship between number of contacts and collision time for CPU based implementation of Method-2

On the GPU, as illustrated in Figure and Figure , the execution time also scales linearly, with Method-2 showing a slight deviation when the number of contacts exceeds 1 million. Note that Method-2 is about 1.5 times faster than Method-1. Figure and **Error! Reference source not found.** report the GPU over CPU speedup for Method-1 and Method-2, respectively. For each method, the GPU execution time is compared to the CPU execution time of the same method. As illustrated in those figures, for large numbers of contacts the obtained speedups are about 140 for the first and 130 for the second method.

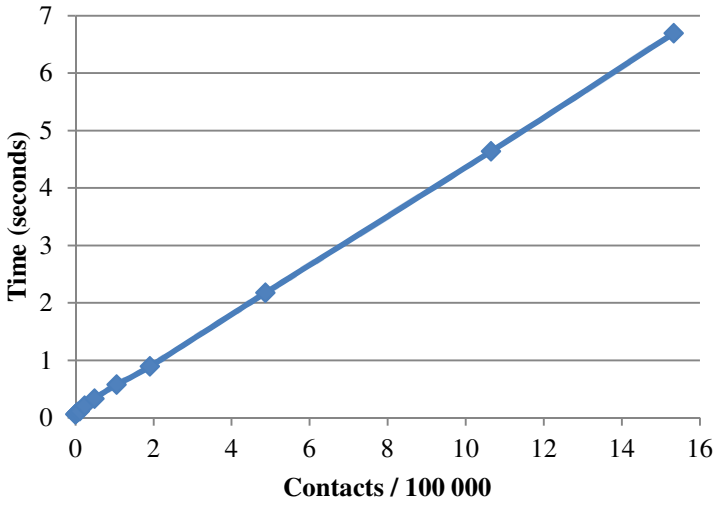


Figure 7 - Relationship between number of contacts and collision time for GPU based implementation of Method-1

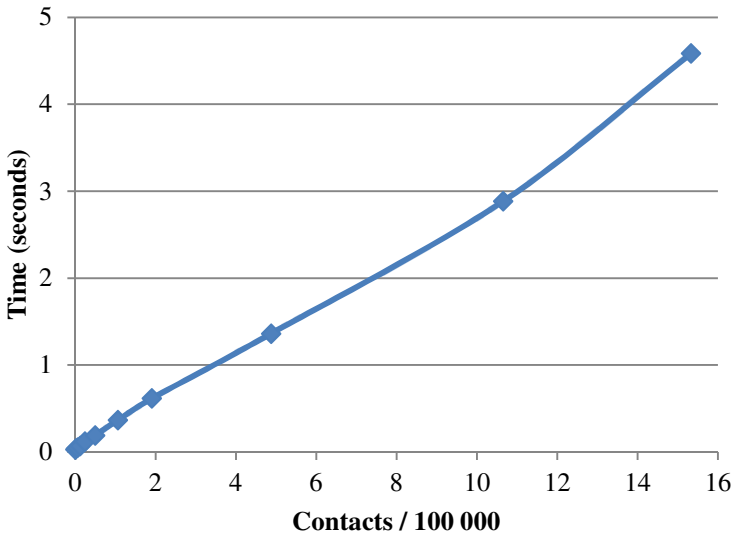


Figure 8 - Relationship between number of contacts and collision time for GPU based implementation of Method-2

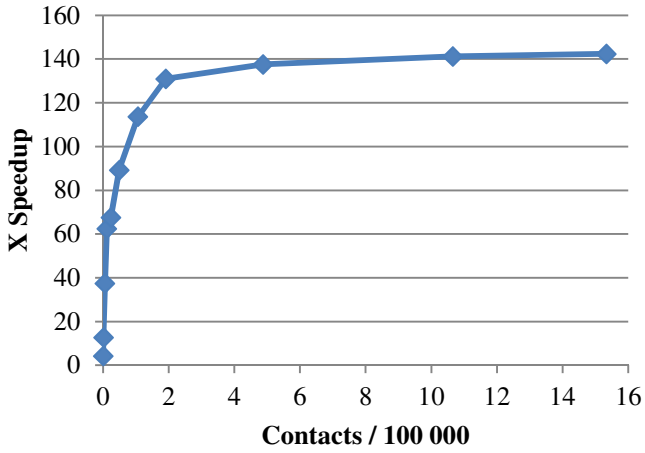


Figure 9 - GPU vs CPU speedup, Method-1

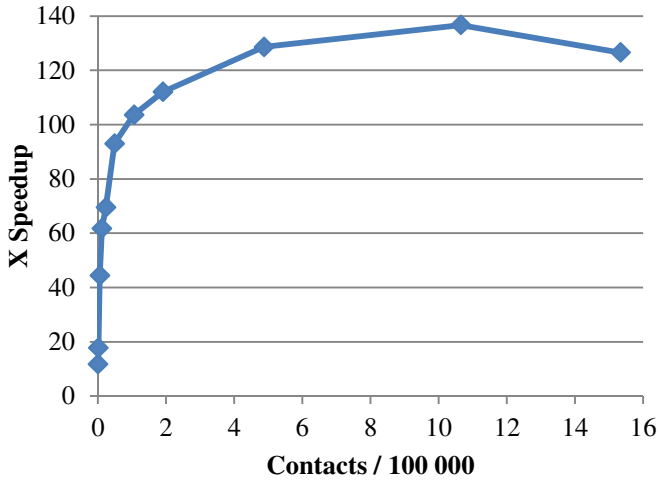


Figure 10 - GPU vs CPU speedup, Method-2

When interpreting the results in Figure and **Error! Reference source not found.** one should keep in mind a point made in [35]: in many cases GPU/CPU speedup gaps can be reduced when the CPU algorithm is tailored and optimized to meet the constraints associated with sequential execution on a x86 architecture. To avoid this pitfall, a numerical experiment was carried out by comparing against BPL, a library optimized to run on x86 architectures and which is the outcome of a community effort to implement an open-source video gaming development package. Since BPL does not natively support an ellipsoid object and instead approximates it as a tessellated object, the performance of the proposed GPU-based ellipsoid contact detection was compared against that of the BPL sphere contact detection solution in its sequential CPU implementation. In this experiment, on the GPU side each ellipsoid had three equal radii. Although the ellipsoid contact detection required a much more compute intensive algorithm, a 14 fold speedup was obtained by the GPU-based ellipsoid contact detection with Method-2 (see **Error! Reference source not found.**).

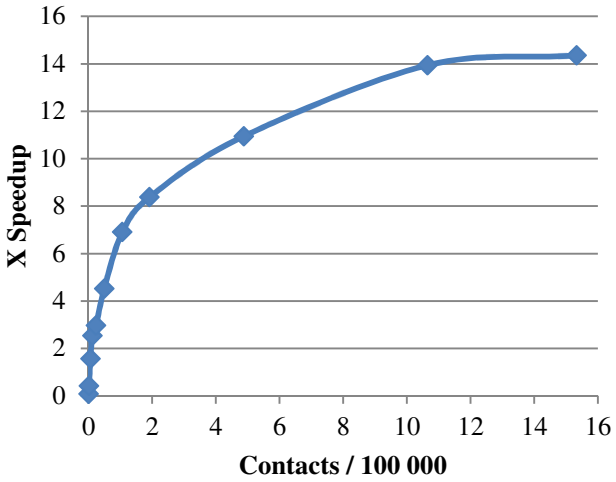


Figure 11 –Speedup of GPU-based ellipsoid contact detection with respect to CPU-based sphere contact detection with BPL

Finally, the time comparison of the proposed parallel ellipsoid contact detection and the parallel sphere contact detection described in [25], both GPU implementations, demonstrated roughly a 14 fold computational overhead (slowdown) for the ellipsoid contact detection.

6. Conclusions

Three ideas combine in this paper to lead to a multi-level parallel ellipsoid collision detection algorithm. The inner level of this algorithm solves an optimization problem for which first and second order analytical sensitivity information is used to produce collision state information (normal vector, penetration, contact points, etc.). At an intermediate level, a new approach based on the characteristic equation associated with the relative position and orientation of two ellipsoids takes advantage of the roots of a third order polynomial to provide a yes/no answer to the question of whether two ellipsoids are in contact. Finally, at the outer level, a spatial decomposition performs a binning of the ellipsoids to determine the pairs of ellipsoids that deserve special attention in the intermediate level of the algorithm. Note that the outer stage itself takes advantage of the inner level for rapid ellipsoid binning.

The primary goal of this effort was to design an algorithm that can be mapped for execution on parallel computing hardware. The resulting three-level method was implemented as a ten stage algorithm executed in parallel on GPU cards. The parallel collision detection is shown to scale linearly with the number of contact events and provide a 130 to 140 fold reduction in simulation time for problems with 10^5-10^6 collisions. The parallel implementation was shown to outperform the standard CPU-based sphere contact detection in Bullet by a factor of 14. This effectively prevents the collision detection from being a simulation bottleneck in approaches based on discrete element or differential variational inequality methods for the analysis of granular flows. Two issues remain to be investigated. First, parallel algorithms to characterize the collision of ellipsoids with other geometric primitives such as boxes and cylinders are needed in order to enable simulations of real life granular dynamics applications such as flow in a silo, mobility of tracked vehicles on granular terrain [14], etc. Second, an MPI/CUDA or MPI/OpenCL implementation remains to be pursued to support scenarios with more than one billion contact events for large scale granular dynamics analysis, where the memory requirements exceed the resources available on a single GPU.

Acknowledgment

Financial support for this project was provided by a National Science Foundation grant NSF-CMMI-0840442, and Function Bay, Inc. The authors would like to thank Naresh Khude, Toby Heyn, and Andrew Seidl for feedback provided on an earlier version of this document.

Appendix A

The equation of an ellipsoid in its local principal reference frame located at its geometric center is

$$\varepsilon: \ (x/r_1)^2 + (y/r_2)^2 + (z/r_3)^2 = 1 \tag{A-1}$$

The coordinates $\mathbf{X} = (X,Y,Z)^T$ in the global reference frame of a point on the surface of the ellipsoid are obtained as

$$\mathbf{X} = \mathbf{A}\mathbf{x} + \mathbf{b}\ , \tag{A-2}$$

where $\mathbf{x} = (x,y,z)^T$ is the representation of the point in the local principal reference frame. The points on the two ellipsoids whose surface normal vectors are parallel to assumed contact direction $\mathbf{c}$ are denoted by $\mathbf{P}_1$ and $\mathbf{P}_2$. If $\mathbf{n}_1$ and $\mathbf{n}_2$ are the surface unit normal vectors for ellipsoids ε_1 and ε_2 respectively, then contact between $\mathbf{P}_1$ and $\mathbf{P}_2$ establishes that

$$\mathbf{n}_1 = \mathbf{c} \ \text{ and } \ \mathbf{n}_2 = -\mathbf{c}\ . \tag{A-3}$$

The normal vectors $\mathbf{n}_1$ and $\mathbf{n}_2$ are expressed in the local reference frame as

$$\mathbf{n}' = \mathbf{A}^T\mathbf{n}, \tag{A-4}$$

Since $\nabla\varepsilon$ is also normal to ε, there exists a scalar λ such that

$$\lambda \nabla \varepsilon = \mathbf{n}', \ \ \lambda > 0 . \tag{A-5}$$

Since

$$\nabla \varepsilon = (2x/r_1^2, 2y/r_2^2, 2z/r_3^2)^T , \tag{A-6}$$

by assuming $\mathbf{n}' = (n_1', n_2', n_3')^T$, it follows that

$$x = r_1^2 n_1'/2\lambda , \ y = r_2^2 n_2'/2\lambda , \ z = r_3^2 n_3'/2\lambda \ . \tag{A-7}$$

Based on Eqs. (A-1) and (A-7) ,

$$\lambda^2 = \frac{1}{4}(r_1^2 n_1'^2 + r_2^2 n_2'^2 + r_3^2 n_3'^2) = \frac{1}{4}\left\|\mathbf{R}\mathbf{n}'\right\|^2 \ , \tag{A-8}$$

where

$$\mathbf{R} \equiv \begin{pmatrix} r_1 & 0 & 0 \\ 0 & r_2 & 0 \\ 0 & 0 & r_3 \end{pmatrix} . \tag{A-9}$$

Equation (A-8) always has a non-zero solution $\lambda > 0$. This equation can be rewritten as,

$$\lambda^2 = \frac{1}{4}\mathbf{n}^T \mathbf{M} \mathbf{n} , \tag{A-10}$$

where

$$\mathbf{M} = \mathbf{A}\mathbf{R}^2\mathbf{A}^T , \tag{A-11}$$

is a symmetric positive definite matrix that does not depend on α_1 and α_2 . Eq. (A-7) is used in conjunction with (A-3), (A-4), and (A-10) to express $\mathbf{x} = (x, y, z)^T$ as a function of $\mathbf{c}$. Accordingly, the global coordinates of associated $\mathbf{P}_1$ and $\mathbf{P}_2$ on ε_1 and ε_2 can be obtained from (A-2) as,

$$\mathbf{x} = \frac{1}{2\lambda}\mathbf{R}^2\mathbf{n}' = \frac{1}{2\lambda}\mathbf{R}^2\mathbf{A}^T\mathbf{n}, \tag{A-12}$$

$$\mathbf{P} = \frac{1}{2\lambda}\mathbf{A}\mathbf{R}^2\mathbf{A}^T\mathbf{n} + \mathbf{b} = \frac{1}{2\lambda}\mathbf{M}\mathbf{c} + \mathbf{b}. \tag{A-13}$$

This implies that $\mathbf{d}$ is written as an explicit function of $\mathbf{c}$:

$$\mathbf{d} = \mathbf{P}_1 - \mathbf{P}_2 = (\frac{1}{2\lambda_1}\mathbf{M}_1 + \frac{1}{2\lambda_2}\mathbf{M}_2)\mathbf{c} + (\mathbf{b}_1 - \mathbf{b}_2). \tag{A-14}$$

The first partial derivatives of $\mathbf{d}$, $\partial \mathbf{d}/\partial \alpha_i$, are obtained from (A-14) and first partial derivatives of $\mathbf{P}$,

$$\frac{\partial \mathbf{P}}{\partial \alpha_i} = \frac{\partial \mathbf{P}}{\partial \mathbf{c}}\frac{\partial \mathbf{c}}{\partial \alpha_i} + \frac{\partial \mathbf{P}}{\partial \lambda}\frac{\partial \lambda}{\partial \alpha_i} = \frac{1}{2\lambda}\mathbf{M}\frac{\partial \mathbf{c}}{\partial \alpha_i} - \frac{1}{2\lambda^2}\mathbf{M}\mathbf{c}\frac{\partial \lambda}{\partial \alpha_i}. \tag{A-15}$$

From Eq. (A-10),

$$\frac{\partial \lambda}{\partial \alpha_i} = \frac{1}{4\lambda}\mathbf{c}^T\mathbf{M}\frac{\partial \mathbf{c}}{\partial \alpha_i}. \tag{A-16}$$

Therefore, the first partial derivative can be written as

$$\frac{\partial \mathbf{P}}{\partial \alpha_i} = (\frac{1}{2\lambda}\mathbf{M} - \frac{1}{8\lambda^3}\mathbf{M}\mathbf{c}\mathbf{c}^T\mathbf{M})\frac{\partial \mathbf{c}}{\partial \alpha_i}. \tag{A-17}$$

Knowing λ, $\frac{\partial \lambda}{\partial \alpha_i}$, $\mathbf{c}$, and $\frac{\partial \mathbf{c}}{\partial \alpha_i}$, the second partial derivatives $\frac{\partial^2 \mathbf{P}}{\partial \alpha_i \partial \alpha_j}$ can be obtained from Eq. (A-17),

$$\frac{\partial^2 \mathbf{P}}{\partial \alpha_i \partial \alpha_j} = \frac{\partial}{\partial \lambda}(\frac{\partial \mathbf{P}}{\partial \alpha_i})\frac{\partial \lambda}{\partial \alpha_j} + \frac{\partial}{\partial \mathbf{c}}(\frac{\partial \mathbf{P}}{\partial \alpha_i})\frac{\partial \mathbf{c}}{\partial \alpha_j} + \frac{\partial}{\partial \mathbf{c}_{\alpha_i}}(\frac{\partial \mathbf{P}}{\partial \alpha_i})\frac{\partial \mathbf{c}_{\alpha_i}}{\partial \alpha_j}, \tag{A-18}$$

where $\mathbf{c}_{\alpha_i} \equiv \partial \mathbf{c}/\partial \alpha_i$. The second partial derivatives assume the form

$$\frac{\partial^2 \mathbf{P}}{\partial \alpha_i \partial \alpha_j} = (-\frac{1}{2\lambda^2}\mathbf{M} + \frac{3}{8\lambda^4}\mathbf{M}\mathbf{c}\mathbf{c}^T\mathbf{M})\frac{\partial \lambda}{\partial \alpha_j}\frac{\partial \mathbf{c}}{\partial \alpha_i} + \frac{\partial}{\partial \mathbf{c}}(\frac{1}{2\lambda}\mathbf{M}\frac{\partial \mathbf{c}}{\partial \alpha_i} - \frac{1}{8\lambda^3}\mathbf{M}\mathbf{c}\mathbf{c}^T\mathbf{M}\frac{\partial \mathbf{c}}{\partial \alpha_i})\frac{\partial \mathbf{c}}{\partial \alpha_j} + (\frac{1}{2\lambda}\mathbf{M} - \frac{1}{8\lambda^3}\mathbf{M}\mathbf{c}\mathbf{c}^T\mathbf{M})\frac{\partial^2 \mathbf{c}}{\partial \alpha_i \partial \alpha_j}. \tag{A-19}$$

The second term in right hand side of equation (A-19) can be written as

$$\frac{\partial}{\partial \mathbf{c}}(\frac{1}{2\lambda}\mathbf{M}\frac{\partial \mathbf{c}}{\partial \alpha_i} - \frac{1}{8\lambda^3}\mathbf{M}\mathbf{c}\mathbf{c}^T\mathbf{M}\frac{\partial \mathbf{c}}{\partial \alpha_i})\frac{\partial \mathbf{c}}{\partial \alpha_j} = -\frac{1}{8\lambda^3}[(\mathbf{c}^T\mathbf{M}\frac{\partial \mathbf{c}}{\partial \alpha_i})\mathbf{M} + \mathbf{M}\mathbf{c}(\frac{\partial \mathbf{c}}{\partial \alpha_i})^T\mathbf{M}]\frac{\partial \mathbf{c}}{\partial \alpha_j}. \tag{A-20}$$

By substituting Eqs. (A-20) and (A-16) into (A-19), the second order partial derivatives can be expressed explicitly in terms of $\mathbf{c}$ and its derivatives

$$\frac{\partial^2 \mathbf{P}}{\partial \alpha_i \partial \alpha_j} = (-\frac{1}{8\lambda^3}\mathbf{M} + \frac{3}{32\lambda^5}\mathbf{M}\mathbf{c}\mathbf{c}^T\mathbf{M})(\mathbf{c}^T\mathbf{M}\frac{\partial \mathbf{c}}{\partial \alpha_j})\frac{\partial \mathbf{c}}{\partial \alpha_i} - \frac{1}{8\lambda^3}[(\mathbf{c}^T\mathbf{M}\frac{\partial \mathbf{c}}{\partial \alpha_i})\mathbf{M} + \mathbf{M}\mathbf{c}(\frac{\partial \mathbf{c}}{\partial \alpha_i})^T\mathbf{M}]\frac{\partial \mathbf{c}}{\partial \alpha_j} + (\frac{1}{2\lambda}\mathbf{M} - \frac{1}{8\lambda^3}\mathbf{M}\mathbf{c}\mathbf{c}^T\mathbf{M})\frac{\partial^2 \mathbf{c}}{\partial \alpha_i \partial \alpha_j}. \tag{A-21}$$

Note that $\partial \mathbf{P}_1/\partial \alpha_i$ and $\partial^2 \mathbf{P}_1/\partial \alpha_i \partial \alpha_j$ can be directly obtained by substituting appropriate matrices in the given equations. However, for $\partial \mathbf{P}_2/\partial \alpha_i$ and $\partial^2 \mathbf{P}_2/\partial \alpha_i \partial \alpha_j$ a negative sign should be applied to all terms of the right hand side since $\mathbf{P}_2$ is associated with $-\mathbf{c}$.

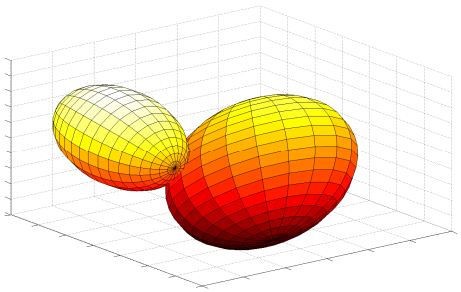
References

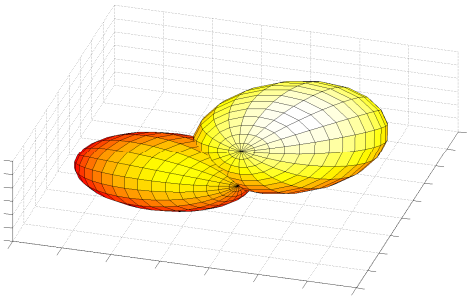
[1] M. Anitescu, A. Tasora, An iterative approach for cone complementarity problems for nonsmooth dynamics, (preprint), Computational Optimization and Applications, (2008).

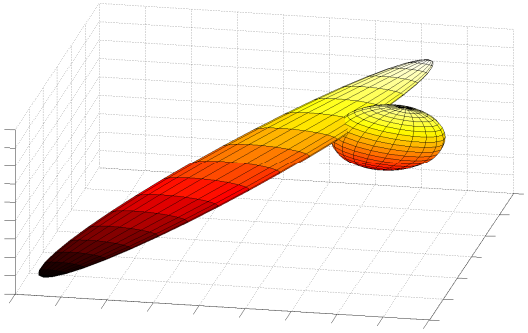
[2] D. Baraff, Dynamic Simulation of Non-Penetrating Rigid Bodies, PhD thesis, Cornell University, 1992.
[3] D. Baraff, An Introduction to Physically Based Modeling:*Rigid Body Simulation II—Nonpenetration Constraints*, in: SIGGRAPH *Course Notes*, 1997.
[4] A.H. Barr, Superquadrics and angle-preserving transformations, IEEE Computer graphics and Applications, 1 (1981) 11-23.
[5] N. Chakraborty, J. Peng, S. Akella, J. Mitchell, Proximity queries between convex objects: An interior point approach for implicit surfaces, IEEE Transactions on Robotics, 24 (2008) 211.
[6] Y. Choi, J. Chang, W. Wang, M. Kim, G. Elber, Continuous collision detection for ellipsoids, IEEE transactions on visualization and computer graphics, 15 (2009) 311-325.
[7] P.W. Cleary, N. Stokes, J. Hurley, Efficient collision detection for three dimensional super-ellipsoidal particles, (1997).
[8] C. Erwin, Physics Simulation Forum, http://www.bulletphysics.com/Bullet/wordpress/, January 15, 2010.
[9] R. Escudie, N. Epstein, J. Grace, H. Bi, Effect of particle shape on liquid-fluidized beds of binary (and ternary) solids mixtures: segregation vs. mixing, Chemical Engineering Science, 61 (2006) 1528-1539.
[10] N. Govindaraju, M. Lin, D. Manocha, Quick-cullide: Fast inter-and intra-object collision culling using graphics hardware, IEEE Virtual Reality, 2005. Proceedings. VR 2005, (2005) 59-66.
[11] N. Govindaraju, M. Lin, D. Manocha, Fast and reliable collision culling using graphics hardware, IEEE Transactions on Visualization and Computer Graphics, 12 (2006) 143-154.
[12] N. Govindaraju, S. Redon, M. Lin, D. Manocha, CULLIDE: Interactive collision detection between complex models in large environments using graphics hardware, in Proceedings of: Eurographics Association, 2003, pp. 25-32.
[13] E. Haug, Computer aided kinematics and dynamics of mechanical systems, Allyn and Bacon Boston, 1989.
[14] T. Heyn, H. Mazhar, D. Negrut, On the simulation of tracked vehicles operating on granular terrain: a parallel multibody dynamics approach (submitted), Multibody System Dynamics, (2010).
[15] J. Hoberock, N. Bell, Thrust, C++ Template Library for CUDA, http://code.google.com/p/thrust/, July 1, 2010.
[16] C. Hogue, Shape representation and contact detection for discrete element simulations of arbitrary geometries, Engineering Computations, 15 (1998) 374-390.
[17] S. Johnson, J.R. Williams, B. Cook, Contact resolution algorithm for an ellipsoid approximation for discrete element modeling, Engineering Computations, 21 (2004) 215-234.
[18] N. Kapre, A. DeHon, Performance comparison of single-precision SPICE Model-Evaluation on FPGA, GPU, Cell, and multi-core processors, in Proceedings of: International Conference on Field Programmable Logic and Applications, 2009, pp. 65-72.
[19] M. Kodam, R. Bharadwaj, J. Curtis, B. Hancock, C. Wassgren, Cylindrical object contact detection for use in discrete element method simulations. Part I-Contact detection algorithms, Chemical Engineering Science, 65 (2010) 5852-5862.
[20] M.R. Kuhn, Smooth convex three-dimensional particle for the discrete-element method, Journal of Engineering Mechanics, 129 (2003) 539.
[21] E. Larsen, S. Gottschalk, M. Lin, D. Manocha, Fast distance queries with rectangular swept sphere volumes, in Proceedings of: Proceedings of IEEE International Conference on Robotics and Automation, 2000, pp. 3719–3726.
[22] S. Le Grand, Broad-Phrase Collision Detection with CUDA, in: H. Nguyen (Ed.) GPU Gems 3, Addison-Wesley, 2008, pp. 697-721.
[23] X. Lin, T. Ng, P. Fellow, Contact detection algorithms for three-dimensional ellipsoids in discrete element modelling, Mechanics of Cohesive-frictional Materials, 19 (1995) 653-659.
[24] E. Lindholm, J. Nickolls, S. Oberman, J. Montrym, NVIDIA Tesla: A Unified Graphics and Computing Architecture, Micro, IEEE, 28 (2008) 39-55.
[25] H. Mazhar, T. Heyn, D. Negrut, A scalable parallel method for large collision detection problems, Multibody System Dynamics, 26 (2011) 37-55.
[26] NVIDIA Corporation, NVIDIA CUDA: Compute Unified Device Architecture, Programming Guide, in, NVIDIA Corporation, Santa Clara, 2008.
[27] NVIDIA Corporation, NVIDIA CUDA Developer Zone, http://developer.nvidia.com/object/cuda_3_1_downloads.html, July 23, 2010.
[28] NVIDIA Corporation., Tesla C2050 and Tesla C2070 Computing Processor Board, http://www.nvidia.com/docs/IO/43395/BD-04983-001_v03.pdf, July 23, 2010.
[29] H. Ouadfel, L. Rothenburg, An algorithm for detecting inter-ellipsoid contacts, Computers and geotechnics, 24 (1999) 245-263.
[30] A.V. Potapov, C.S. Campbell, A fast model for the simulation of non-round particles, Granular Matter, 1 (1998) 9-14.
[31] N. Satish, M. Harris, M. Garland, Designing efficient sorting algorithms for manycore GPUs, NVIDIA Technical Report NVR-2008-001, (2008).
[32] A. Tasora, D. Negrut, M. Anitescu, A GPU-based approach for simulating rigid body dynamics with frictional contact, Journal of Multi-Body Dynamics, 222(K4) (2008) 315-326.
[33] E. Tijskens, J.D. Baerdemaeker, H. Ramon, Strategies for contact resolution of level surfaces, Engineering Computations, 21 (2004) 137-150.

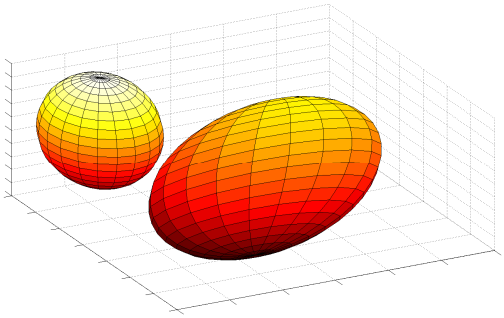
[34] J.M. Ting, M. Khwaja, L.R. Meachum, J.D. Rowell, An ellipse-based discrete element model for granular materials, Mechanics of Cohesive-frictional Materials, 17 (1993) 603-623.
[35] R. Vuduc, A. Chandramowlishwaran, J. Choi, M. Guney, A. Shringarpure, On the limits of GPU acceleration, in: Proceedings of the 2nd USENIX conference on Hot topics in parallelism, USENIX Association, Berkeley, CA, 2010.
[36] C.Y. Wang, C.F. Wang, J. Sheng, A packing generation scheme for the granular assemblies with 3D ellipsoidal particles, Mechanics of Cohesive-frictional Materials, 23 (1999) 815-828.
[37] W. Wang, J. Wang, M.S. Kim, An algebraic condition for the separation of two ellipsoids, Computer aided geometric design, 18 (2001) 531-539.
[38] C. Wellmann, C. Lillie, P. Wriggers, A contact detection algorithm for superellipsoids based on the common-normal concept, Engineering Computations, 25 (2008) 432-442.
[39] J.R. Williams, R. O'Connor, A linear complexity intersection algorithm for discrete element simulation of arbitrary geometries, Engineering Computations, 12 (1995) 185-202.
[40] J.R. Williams, A.P. Pentland, Superquadrics and modal dynamics for discrete elements in interactive design, Engineering Computations, 9 (1992) 115-127.
[41] S. Wright, J. Nocedal, Numerical optimization, second ed., Springer, Berlin, 2006.
[42] C. Xu, J. Zhu, Parametric study of fine particle fluidization under mechanical vibration, Powder Technology, 161 (2006) 135-144.
[43] X. Zhang, L. Vu-Quoc, Simulation of chute flow of soybeans using an improved tangential force-displacement model, Mechanics of materials, 32 (2000) 115-129.

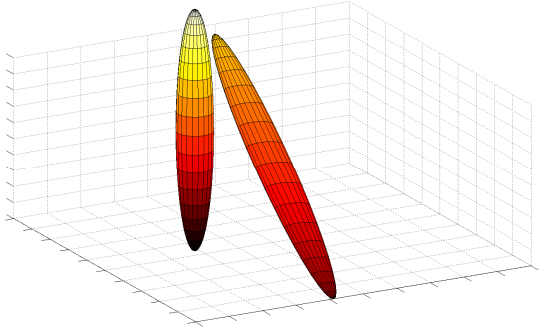
Extras











Movie

http://vimeo.com/31810270