

Physical Modelling  
using Geometric Algebra  
by  
Jonathan Cameron (SEL)  
Fourth-year undergraduate project in  
Group E, 2002/2003

I hereby declare that, except where specifically indicated, the work submitted herein is my own original work.

Signed: \_\_\_\_\_ Date: \_\_\_\_\_

## Summary

The vast majority of simulations of physical systems, including the movement of rigid bodies under forces, are carried out using vector algebra. Some aspects of this approach are unintuitive and hence can prove difficult to implement. Geometric Algebra (GA) may overcome these problems. This will give advantages in the speed of implementation of any new simulation.

Further advances are made by utilising the mapping from  $\mathbb{R}^3$  to  $\mathbb{R}^5$ , introduced by Hestenes, here referred to as the Conformal Model (CGA). CGA provides simple representations of geometric entities (lines, circles, planes and spheres) and allows for their relatively easy manipulation and intersection. This provides considerable simplifications, when applied to the problem of collision detection and response.

The project builds upon the work and code of R. Wareham, relating to the intersections of the geometric entities, to set a convention for defining solid spheres and utilise this in the creation of bounding spheres for arbitrary objects. This is accomplished via an implementation of the geometric functions required by Welzl's Algorithm. These bounding spheres are then utilised to reduce the number of possible collisions to be investigated. Exhaustive collision detection and response between convex bodies is implemented to evaluate the issue of whether GA/CGA provide any advantages within this field.

A substantial portion of the time available was spent gaining an appreciation of the techniques currently employed. This provided insights into possible applications of GA/CGA and a base level against which to evaluate the GA/CGA approaches. The understanding gained allowed the development of a number of tools necessary for the latter stages of the project. These tools included advanced differential equation solvers and algorithms for rapid matrix manipulation. The resulting tool-set was used in the development of a particle and rigid body simulator using conventional methods. The simulator models the motion of point masses under various simple positional constraints and forces. The motion of general rigid bodies moving under simple forces has also been implemented. A GA / CGA based rigid body simulator was also developed. The GA / CGA simulator both allowed the collision detection / response techniques to be evaluated for convex bodies in a realistic test environment and the applications of GA to be evaluated for use in the physical simulations.

---

Several extensions to the project, in which CGA may prove useful, are discussed. These include the construction and use of Sphere-Trees and their inherent link to Voronoi diagrams. Some consideration is also given to the structure of a simulator and methods for reducing the memory usage so as to increase speed when simulations involving many entities are created.

# Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                            | <b>4</b>  |
| 1.1      | Motivation for the Project . . . . .           | 4         |
| 1.2      | Introduction to Geometric Algebra . . . . .    | 5         |
| 1.3      | Conformal Geometric Algebra . . . . .          | 9         |
| 1.4      | Rotors . . . . .                               | 10        |
| 1.5      | Representation of Geometric Entities . . . . . | 12        |
| 1.6      | Intersection of Geometric Entities . . . . .   | 12        |
| 1.7      | Reflection in Geometric Entities . . . . .     | 14        |
| <b>2</b> | <b>Physical Modelling</b>                      | <b>14</b> |
| 2.1      | Particle Dynamics . . . . .                    | 14        |
| 2.2      | Why a GA method was not considered . . . . .   | 19        |
| 2.3      | Rigid Body Dynamics . . . . .                  | 20        |
| 2.4      | Collision Detection . . . . .                  | 25        |
| 2.5      | Collision Response . . . . .                   | 36        |
| 2.6      | CGA / GA versus Traditional . . . . .          | 38        |
| <b>3</b> | <b>Conclusions</b>                             | <b>39</b> |
| <b>4</b> | <b>Future Work</b>                             | <b>40</b> |
| 4.1      | Hardware Implementation . . . . .              | 40        |
| 4.2      | Advanced Collision Detection . . . . .         | 41        |
| 4.3      | Finding Time of Collision . . . . .            | 43        |
| <b>A</b> | <b>Programming Methodology</b>                 | <b>44</b> |
| <b>B</b> | <b>Differential Equation Solvers</b>           | <b>45</b> |
| B.1      | 1st Order Euler Method . . . . .               | 46        |
| B.2      | Runge-Kutta Methods . . . . .                  | 47        |
| B.3      | Adapting the Time Step . . . . .               | 47        |

Figure 1: Physical Modelling within the CG Film Industry

(a) Hair modelled as linked series of Particles.<sup>2</sup>

(b) Cloth animation requires combination of particle, solid and collision modelling.

## 1 Introduction

### 1.1 Motivation for the Project

*To discover whether the use of Geometric Algebra provides any advantages, in terms of coding simplicity or speed, within the creation of a system for the modelling of physical systems.*

The physical systems that may benefit from these mathematical tools are wide ranging. This project will concentrate on only a limited subset, the motion of particles and rigid bodies, moving under simple forces and constraints.

An enormous number of applications make use of such modelling of simple object dynamics. These range from uses within the growing Computer Graphics (CG) based film industry, to the simulation of human movement for use in the removal of ambiguities in motion capture data [5].

Within the film industry the rapid production of realistic<sup>3</sup> object motion is be-

---

<sup>2</sup>Aki Ross from Final Fantasy - note that the rendering of this character's hair is reported to have required 50% of the rendering time required for the whole film - getting it right is vital.

<sup>3</sup>With CG Films it is not always perfect modelling of a system that is required, but instead the appearance of realistic motion. This allows a greater degree of approximation to be used than in traditional engineering simulation.

coming increasingly common as sections<sup>4</sup> of live action are replaced with computer graphics. Real time results, whilst not required to produce CG sequences, do allow a far greater degree of directorial input, vital as these artificial sequences become more complex. Examples of such modelling are given in Fig. 1. Often the final rendering of a sequence may take a number of days so it is important to get it right as early as possible through the use of ‘rapid prototyping’ of a sequence. This is another area where the speed of implementation of a desired effect can become important.

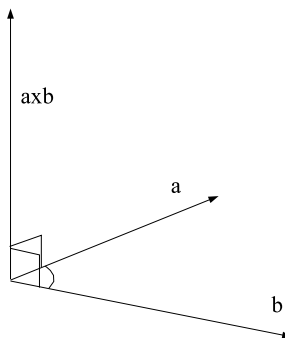
The form of Geometric Algebra that looks likely to offer the most advantages is the conformal model with its simple representation of common geometric shapes (See 1.5).

## 1.2 Introduction to Geometric Algebra

Geometric Algebra (GA) provide a universal language for the description of physical systems. GA has been applied to diverse problems ranging from the physics of black holes to computer vision. The properties and use of GA are best explained by outlining the 3 simplest products.

### 1.2.1 The Outer Product

Figure 2: The Cross Product



In 3D vector algebra the cross product is denoted  $a \times b$ , where  $a$  and  $b$  are vectors (Fig. 2.) This is used frequently within a traditional representation of simple

<sup>4</sup>Entire CG films are now being produced with each successive generation approaching, ever closer, completely believable realism.

mechanics to represent such quantities as angular velocity through the use of a vector defined perpendicular to the plane in which the rotation lies (an axial vector). Intuitively this makes little sense. Why should such quantities be represented using the same type of element as linear velocity, given the two types of velocity are clearly not directly equivalent?

A more intuitive representation is given by a bivector. The bivector can be thought of as a directed area (Fig. 4(a)). To form these bivectors we define the *outer product*, denoted  $a \wedge b$ . The magnitude of this area is the same as that for the standard cross product ( $|a \wedge b| = |a||b| \sin \theta$ , where  $\theta$  is the angle between the two vectors). It is interesting to note that this outer product is defined for any number of space dimensions whilst the cross product only has meaning in 3D. As this element is 2 dimensional it is described as being of *Grade 2*. Note that if we were to take the outer product of a bivector and a vector then we will generate a ‘directed volume’ element. This a Grade 3 element and as with the bivector formed above it only contains elements of single grade. All elements of only single grade are described as *Blades*.

Figure 3: Multidimensional Elements of GA



(a) The Outer Product - Bivector

(b) The Outer Product - Trivector

### 1.2.2 The Inner Product

The inner (or dot) product of vectors  $a$  and  $b$  is denoted  $a \cdot b$ . In GA its purpose is very similar to the equivalent vector algebra operator, in that it describes the proportion of one vector in the direction of another. This can be equally well applied to higher grade elements and is extremely useful for establishing relative positions of objects when used with Conformal Geometric Algebra.

### 1.2.3 The Geometric Product

William Clifford (1845 - 1879) added a third product that unifies the Inner and Outer products. This is known as the Geometric Product and as written  $ab$  where,

$$ab = a \cdot b + a \wedge b \quad (1)$$

Clearly, this object is the sum of a bivector and a scalar. This may appear unintuitive, but consider the definition of complex numbers as the sum of two separate entities that exist in different spaces. In a similar way, a multivector, as multiple grade elements of GA are termed, simply keeps track of the various grades of element produced by the Geometric Product<sup>5</sup>. An axiomatic derivation of GA involves defining the other two products above in terms of the geometric product, which is the fundamental operator of GA [8].

### 1.2.4 Rotation

An important operation in any manipulation of geometrical entities, is that of rotation. In GA rotations are defined by a multivector  $R$  (called a rotor) with only even grades

$$x' = Rx\tilde{R} \quad (2)$$

Consider 3 orthonormal basis vectors of  $\mathbb{R}^3$ ,  $\{e_1, e_2, e_3\}$  and take the geometric<sup>6</sup> products of pairs of these basis vectors to form 3 bivectors,

$$B_1 = e_1e_2, \quad B_2 = e_2e_3, \quad B_3 = e_3e_1 \quad (3)$$

Note that

$$e_i^2 = +1 \quad \forall i = 1, 2, 3 \quad (4)$$

---

<sup>5</sup>In fact it can be easily shown that in 2D the grade 2 element is equivalent to the imaginary part of a complex number [8].

<sup>6</sup>Note that, as the basis vectors are orthonormal, the geometric product of any pair of basis vectors, is equivalent to the outer product of the pair and hence defines a bivector.

Now consider the effect of multiplying  $e_1$  or  $e_1 + e_2$  by  $B_1$ :

$$\begin{aligned} e_1 B_1 &= e_1 e_1 e_2 = e_1^2 e_2 = e_2 \\ (e_1 + e_2) B_1 &= e_1 e_1 e_2 + e_2 e_1 e_2 = e_2 - e_1 \end{aligned}$$

From these examples it is clear that post-multiplying by  $B_1$  has the effect of rotating the vectors counter clockwise by  $90^\circ$ . By using these results and considering a general vector in the plane of magnitude  $r$  and direction  $\theta$ .

$$\begin{aligned} Z = r e^{i\theta} &= r(e_1 \cos \theta + e_2 \sin \theta) \\ &= e_1 r(\cos \theta + B_1 \sin \theta) = e_1 r e^{B_1 \theta} \end{aligned} \quad (5)$$

This equation shows that any vector in the plane defined by  $B_1$  can be expressed as  $e_1$ , scaled by  $r$  and rotated using the multivector  $B_1$ . Now extend this argument to rotations by an angle  $\phi$  in the plane of  $B_1$ .

$$Z' = Z e^{B_1 \phi} = Z(\cos \phi + B_1 \sin \phi) \quad (6)$$

Although the illustrative case is that of the 2D plane, the derivation has at no point assumed this and  $B_1$  can be an arbitrary bivector defining any plane of rotation. Until now only the case of a vector lying in the plane of rotation has been considered. To handle vectors with components out of this plane we decompose the vector into a component within the plane  $x_{\parallel}$  and one perpendicular to the plane  $x_{\perp}$ . If we now consider,

$$\begin{aligned} e^{-B_1 \phi/2} x e^{B_1 \phi/2} &= \left( \cos \frac{\phi}{2} - B_1 \sin \frac{\phi}{2} \right) (x_{\parallel} + x_{\perp}) \left( \cos \frac{\phi}{2} + B_1 \sin \frac{\phi}{2} \right) \\ &= x_{\parallel} e^{B_1 \phi} + x_{\perp} \end{aligned} \quad (7)$$

we see that this form rotates the component in the plane whilst leaving the component out of the plane unaffected. This transform is therefore given by,

$$x \mapsto R x R^{-1} \quad (8)$$

where the rotor  $R = e^{-B\phi/2}$ . Note that we can avoid the necessity for directly inverting  $R$  by defining the reversion of an n-vector  $X = e_i e_j \dots e_k$  as  $\tilde{X} = e_k \dots e_j e_i$

i.e literally reverse the components. Now observe from the definition for  $R$  that  $\tilde{R} \equiv R^{-1}$ .

### 1.3 Conformal Geometric Algebra

Conformal Geometric Algebra (CGA) has been used in several key areas of this project due to the advantages provided by CGA's simple and elegant representation and manipulation of geometric elements.

The 3D Geometric Algebra ( $\mathbb{R}^3$ ) described above is extended by adding a pair of elements to the 3 existing base vectors. Clearly these elements also result in additional bivectors and trivectors. It is interesting to note that now we have 5 basis vectors, we can form grade 4 and 5 elements. The resulting basis vectors are defined as,

$$X = \{e_1, e_2, e_3, e, \bar{e}\}$$

where the two additional basis vectors are defined such that,

$$\begin{aligned} e^2 &= +1, & \bar{e}^2 &= -1, & e \cdot \bar{e} &= 0 \\ e \cdot e_i &= \bar{e} \cdot e_i = 0 & \forall i &\in \{1, 2, 3\} \end{aligned} \quad (9)$$

To convert a 3D coordinate  $\bar{x} = \{x_1, x_2, x_3\}$  the following transform is used:

$$X = \frac{1}{2}(\bar{x}^2 n + 2\bar{x} - \bar{n}) \quad (10)$$

where<sup>7</sup>,

$$n = e + \bar{e}, \quad \bar{n} = e - \bar{e}$$

Usefully, the first 3 elements of the resulting vector (now in  $\mathbb{R}^5$ ) are the same as the 3 coordinates (x,y,z) and thus can be easily extracted. This means that conversion between types only has an overhead in one direction. As a result, many intermediate results and elements describing states are stored and manipulated using CGA to avoid the need for the expensive conversion. Now consider the two

---

<sup>7</sup>It can be shown that  $n$  can be associated with the point at infinity and  $\bar{n}$  the origin [7].

vectors  $n$  and  $\bar{n}$ . Note that both are null vectors since,

$$\begin{aligned} n^2 &= (e + \bar{e}) \cdot (e + \bar{e}) = e^2 + 2e \cdot \bar{e} + \bar{e}^2 = 0 \\ \bar{n}^2 &= (e - \bar{e}) \cdot (e - \bar{e}) = e^2 - 2e \cdot \bar{e} + \bar{e}^2 = 0 \end{aligned} \quad (11)$$

If the mapping into CGA given above is considered it can be shown to map 3D vectors into null vectors in the CGA.<sup>8</sup>

## 1.4 Rotors

In CGA rotors can represent a lot more than a simple rotation. An interesting property of general rotors is that they can be easily interpolated. Although no use has been made of this fact here, one possible application is considered in 4.3.2. In this project the following rotor forms have been used.

### 1.4.1 Rotation

A useful property of the CGA mapping is that pure rotation rotors for simple GA are unaffected and so retain their properties.

$$\begin{aligned} RF(X)\tilde{R} &= \frac{1}{2}R(\bar{x}^2n + 2\bar{x} - \bar{n})\tilde{R} \\ &= \frac{1}{2}(x^2Rn\tilde{R} + 2Rx\tilde{R} - R\bar{n}\tilde{R}) \\ &= F(Rx\tilde{R}) \end{aligned} \quad (12)$$

This result uses the following, easily proved results.

$$Rn\tilde{R} = n, \quad R\bar{n}\tilde{R} = \bar{n} \quad (13)$$

---

<sup>8</sup>The mapping to null vectors is easily shown by considering  $F(x)^2$ , which is equal to 0.

### 1.4.2 Dilation

A rotor  $D_\alpha$  of the form  $e^{\frac{\alpha}{2}e\bar{e}}$  has the effect of dilating by a factor of  $e^{-\alpha}$  (see [7]). This can be shown by,

$$\begin{aligned}
D_\alpha F(X)\tilde{D}_\alpha &= e^{\frac{\alpha}{2}e\bar{e}}\frac{1}{2}(x^2n + 2x - \bar{n})e^{-\frac{\alpha}{2}e\bar{e}} \\
&= \frac{1}{2}(x^2e^{\alpha e\bar{e}}n + 2x - e^{\alpha e\bar{e}}\bar{n}) \\
&= \frac{1}{2}(x^2e^{-\alpha}n + 2x - e^\alpha\bar{n}) \\
&= e^\alpha\frac{1}{2}((e^{-\alpha}x)^2n + 2(e^{-\alpha}x) - \bar{n}) \\
&= e^\alpha F(e^{-\alpha}x)
\end{aligned} \tag{14}$$

Note that the above steps can be verified using a Taylor expansion of the rotor. The dilation rotor is only used within the setup of scenes where it provides a simple method of scaling predefined objects.

### 1.4.3 Translation

Again we consider a different form of rotor, in this case  $T_\alpha = e^{\frac{na}{2}}$ . If we take the Taylor expansion of this rotor,

$$T_\alpha = e^{\frac{na}{2}} = 1 + \frac{na}{2} + \frac{1}{2}\left(\frac{na}{2}\right)^2 \cdots = 1 + \frac{na}{2} \tag{15}$$

since  $n$  is null (therefore  $n^2 = 0$  and  $an = -na$ ). Now consider the effect of the rotor on  $n$ ,  $\bar{n}$  and  $x$ .

$$\begin{aligned}
T_\alpha n \tilde{T}_\alpha &= \left(1 + \frac{na}{2}\right)n\left(1 + \frac{an}{2}\right) \\
&= n + \frac{1}{2}nan + \frac{1}{2}nan + \frac{1}{4}nana \\
&= n
\end{aligned}$$

similarly

$$\begin{aligned}
T_\alpha \bar{n} \tilde{T}_\alpha &= \bar{n} - 2a - a^2n \\
T_\alpha x \tilde{T}_\alpha &= x + n(a \cdot x)
\end{aligned}$$

From this the effect of this rotor on  $F(x)$  can be established,

$$T_\alpha F(x) \tilde{T}_\alpha = F(x + a) \quad (16)$$

Multiple rotors can be combined simply by applying one after another, or by first forming the combined rotor and its inversion.

## 1.5 Representation of Geometric Entities

A crucial feature of CGA within the problems tackled in this project, lies in the simple representation of certain geometric bodies and the ease with which these can be created and then manipulated. The form and some of the properties of these entities are given in Table 1.

## 1.6 Intersection of Geometric Entities

As shown in Table 1 lines, circles, spheres and planes can all be represented as  $X \wedge M$  where  $M$  is a blade of required grade. To consider intersections note,

$$X \wedge M_r = X \wedge M_s = 0 \quad (17)$$

which can be shown to be equivalent to<sup>9</sup>

$$X \wedge [\langle M_r M_s \rangle_{10-r-s}]^* = 0 \quad (18)$$

where  $[ ]^*$  is the dual operator (multiplication by the pseudoscalar<sup>10</sup>,  $r$  and  $s$  are the grades of the two blades being intersected and  $\langle X \rangle_i$  denotes the extraction of the  $i$ -grade component of  $X$ . From the above, the meet operator is defined as

$$M_r \vee M_s = [\langle M_r M_s \rangle_{10-r-s}]^* \quad (19)$$

This defines another single grade entity which can be interpreted as the intersections of the two geometric objects being considered. A more detailed description

---

<sup>9</sup>This version of the meet operator is only applicable to the 3D space in which I am working - by using twice the dimension of the space instead of the 10 in this formula, other size spaces can be considered.

<sup>10</sup>The pseudoscalar is the grade 5 element of the algebra.

Table 1: CGA representation of Geometric entities

| Entity | Defining Blade $M (X \wedge M = 0)$ <sup>a</sup> | Extractable Parameters  | Uses Within Project  |
|--------|--|---|--|
| Line   | $P_1 \wedge P_2 \wedge n$                        |   | Definition of Edges for use with edge /facet intersections (See 2.4.4)       |
| Circle | $P_1 \wedge P_2 \wedge P_3$                      | $M^*$ encodes the radius and centre of the circle <sup>b,c</sup> . Centre easily found by reflecting $n$ in the circles $MnM$ . | Used for evaluation of the depth of collision between Spheres                |
| Plane  | $P_1 \wedge P_2 \wedge P_3 \wedge n$             |   | Edge facet intersection requires intersection of segment of line with plane. |
| Sphere | $P_1 \wedge P_2 \wedge P_3 \wedge P_4$           | As with circle $M^*$ encodes radius and centre. Centre easily found by reflecting $n$ in sphere. <sup>d</sup>                   | Bounding Spheres - the first level of collision detection (See 2.4.1)        |

<sup>a</sup> $P_i$  is a point on the given entity.<sup>b</sup>See [7] for details on the extraction of these parameters.<sup>c</sup> $M^*$  denotes the dual of  $M$ .<sup>d</sup>An interesting point is that this simple form does not specify the direction of normals - a convention is need for checking whether points lie within the circle and this will be considered later.

the meet operator can be found in [7].

## 1.7 Reflection in Geometric Entities

A useful result of CGA is that reflections in a given entity, be it a line, plane, circle, or sphere can be performed by ‘sandwiching’ the object to be reflected with that in which it is being reflected. The results are most easily explained for a point reflected in various bodies. The object is again defined as  $X \wedge M = 0$  and the point by a 1-blade  $c$ . The reflected point is then given by:

$$c' = McM \tag{20}$$

This method is use in a number of places within this project as a means of discovering the distance of one object form another. For example  $c' = LcL$  where L is a 3-blade defining a line gives us a means of finding the nearest point on the line to a point - simply find the midpoint of  $c$  and  $c'$ .

Another use of reflection concerns  $n$ , the point at infinity. If we reflect  $n$  in either a sphere or a circle then we immediately establish the centre point.

## 2 Physical Modelling

### 2.1 Particle Dynamics

In order to build up both the extensive tool set needed for more complex modelling and to gain experience in the use of the relevant mathematics within an easily understood field, the simulation of particles was considered prior to the more complex simulation of rigid bodies.

#### 2.1.1 Unconstrained Dynamics

The simplest system, and hence the first to be implemented is that of modelling point masses acting under various forces<sup>11</sup>. The changing state of such a system is dependent upon initial conditions and the force acting. These are related by

---

<sup>11</sup>This stage was completed in parallel with development of the CGA and GA libraries used later in the project.

Newton's 2<sup>nd</sup> Law:

$$F = ma \quad (21)$$

A crucial element of all fast computational modelling of differential equation controlled systems, is that direct solution of a general 2nd order differential equation is extremely difficult. To avoid this the 2nd order equation is expressed as a pair of linked 1st order equations:

$$\begin{bmatrix} \ddot{X} \\ \dot{X} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{X} \\ X \end{bmatrix} + M^{-1} \begin{bmatrix} F \\ 0 \end{bmatrix} \quad (22)$$

Note that in the above equations, X represents a 3 dimensional vector containing all three coordinates of the particle. To use a general differential equation solver, it is necessary to form the large matrices and vectors giving a linked set of  $2n'$  differential equations which can then be solved simultaneously.

The program used C++'s inheritance system to allow a vector<sup>12</sup> containing pointers to all of the different instances of the force classes. Each force was then asked to calculate its own contribution to the overall force on a given particle. The simple forces implemented included are shown in table (2)

Table 2: Forces implemented for Particle Motion Simulator

| Force                                       | Equation                               |
|---|--|
| Inverse square attraction between particles | $\frac{Cm_1m_2}{ r_1-r_2 ^2}$          |
| Inverse square attraction to a point        | $\frac{Dm}{r_1^2}$                     |
| Simple Directional Force                    | $mg$                                   |
| Viscous Drag (Linear)                       | $-k\dot{\mathbf{x}}$                   |
| Viscous Drag (Quadratic)                    | $-k \dot{\mathbf{x}} \dot{\mathbf{x}}$ |

To ensure maximum flexibility the differential solvers coded at this point in the project were built to be completely generic. The solver is simply passed a 'differential function' to generate the required differentials and an accuracy function

<sup>12</sup>Vector as defined in the C++ Standard Template Library STL.

to dictate the required accuracy for each element when variable time-step solvers are used. A fuller consideration of the solvers implemented can be found in Appendix B.

### 2.1.2 Constrained Dynamics

The addition of constraints to the model is accomplished through the use of additional forces. These forces are required to ensure that no specified constraint is broken, i.e. the constraint forces convert the particles' accelerations into 'legal' accelerations, consistent with the constraints. A simple example of this, a single pendulum, was given in the Technical Milestone Report [3]. Here I will instead derive the general equations for handling constraints and consider the form of several such constraints<sup>13</sup>.

**General Constraint Force Calculation** Unfortunately the analytical approach used for the simple case of a pendulum as given in [3] does not easily scale up to more complex systems. Thus, an alternative is needed that allows the same principles to be applied to arbitrary constraints. To do this the required states are grouped into a state vector ( $q$ ) and a mass matrix ( $M$ ) and force vector ( $Q$ ) are formed. As the state vector contains three elements for each particle ( $x, y, z$  coordinates) the mass matrix is a square diagonal matrix with elements ( $m_1, m_1, m_1, m_2, m_2, m_2, m_3, \dots$ ) on the diagonal. The global equation (from Newton's 2nd Law) is therefore

$$\ddot{q} = M^{-1}Q_{tot}$$

The constraints must also be handled in a global fashion and so the scalar constraint functions are concatenated into the vector function  $C(q)$ . In order to maintain the constraints (i.e.  $c = 0$ ), it is clear that  $\dot{c} = 0$  and again the problem to solve is finding the constraint force  $\hat{Q}$  that, when added to the applied force  $Q$  to give  $Q_{tot}$ , ensures  $\ddot{C} = 0$ . To begin, take derivatives of  $C$ . In the simple case of the pendulum [3], we had a specific expression for the constraint function and so could easily find the derivatives. This time round we are dealing with the general case and so these

---

<sup>13</sup>All derivations based upon [14].

cannot be evaluated at this stage. From the chain rule

$$\dot{C} = \frac{\delta C}{\delta q} \dot{q}$$

From here on we will denote  $\delta C/\delta q$  as  $J$  as it is the *Jacobian* of  $C$ .

Differentiating again with respect to time gives

$$\ddot{C} = \dot{J}\dot{q} + J\ddot{q}$$

Rather than directly differentiating  $J$  using the chain rule  $\dot{J} = \delta J/\delta q\dot{q}$  which would result at an intermediate stage in a rank 3 tensor (which is hard to represent efficiently as well as visualize) we use the fact that  $\dot{J} = \delta\dot{C}/\delta q$  which results in a simple matrix.

Next we use the equation of motion to eliminate  $\ddot{q}$  giving

$$\begin{aligned} \ddot{C} &= \dot{J}\dot{q} + JM^{-1}(Q + \hat{Q}) = 0 \\ JM^{-1}\hat{Q} &= -\dot{J}\dot{q} - JM^{-1}Q \end{aligned} \quad (23)$$

When thinking about the above equation it helps to consider  $J$  as a collection of vectors, each of which is the gradient of one of the scalar constraints comprising  $C$ . Since we require that  $C = 0$  these vectors must be normals to the hypersurfaces representing the state-space directions in which the system is not permitted to move. Vectors of the form  $J^T\lambda'$  are linear combinations of these gradient vectors and hence span the set of prohibited directions. Restricting the constraint force to this space therefore ensures that no energy change of the system will be caused by the constraint forces. As with the case of the pendulum, the constraint force must result in no energy changes. This will lead to a more complex situation when dealing with rigid bodies.

To find  $\lambda$  we substitute for  $\hat{Q}$  into equation(23) to give

$$JM^{-1}J^T\lambda = -\dot{J}\dot{q} - JM^{-1}Q \quad (24)$$

$JM^{-1}J^T$  is a square matrix with the dimension of  $C$ . Once  $\lambda$  has been obtained it is multiplied by  $J^T$  to obtain  $\hat{Q}$  and we can then proceed as before.

The actual computation of the constraint forces using the above equation is

implemented so as to take full advantage of the sparse nature of the constraints matrix. Principally a conjugate gradient solver<sup>14</sup> based upon the approach given in Numerical Recipes [10] is used to obtain the Lagrange multiplier vector,  $\lambda$ . The methods used here closely follow those in [14] and will not be explained fully as they only standard computational techniques are used.

The simplest constraint (Type 1) to implement is that of a particle constrained to remain a fixed distance,  $d$ , from a point  $X_n$ . Note that the vectors  $J$  and  $\dot{J}$  simply represent the relevant elements of the full matrices.

$$\begin{aligned}
 C &= 0.5((X_1 - X_n) \cdot (X_1 - X_n) - d^2) \\
 \dot{C} &= (\dot{X}_1) \cdot (X_1 - X_n) \\
 J &= (X_1 - X_n) \\
 \dot{J} &= (\dot{X}_1)
 \end{aligned} \tag{25}$$

The other form of constraint (Type 2) implemented fixes the distance,  $d$ , between two particles

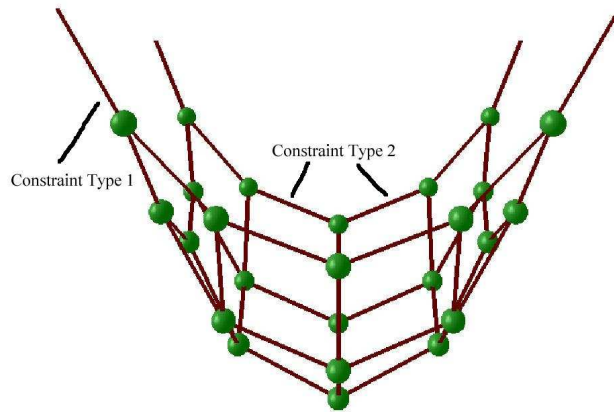
$$\begin{aligned}
 C &= 0.5((X_1 - X_2) \cdot (X_1 - X_2) - d^2) \\
 \dot{C} &= (\dot{X}_1 - \dot{X}_2) \cdot (X_1 - X_2) \\
 J_1 &= (X_1 - X_2) \\
 J_2 &= -(X_1 - X_2) \\
 \dot{J}_1 &= (\dot{X}_1 - \dot{X}_2) \\
 \dot{J}_2 &= -(\dot{X}_1 - \dot{X}_2)
 \end{aligned} \tag{26}$$

Potentially any analytic function for which we can evaluate the required differentials can be used as a constraint, providing a way of constraining particles to fixed paths, a useful technique in building complex animation sequences. A number of examples of this constraint's system in action can be found on the attached CD. One of the more complex examples implemented is the hanging net in Fig. 2.1.2. This clearly illustrates the two types of constraint implemented.

---

<sup>14</sup>A conjugate gradient method is used to exploit the sparsity and symmetry of the matrix. Recent research has shown that this particular algorithm can be run extremely fast on the latest GPU - graphics processor units.

Figure 4: The Particle Net at end of simulation - See [3] for details more images or simulation. Attached CD includes video sequences of this model under various conditions.



## 2.2 Why a GA method was not considered

From [4] it is clear that there are very few differences in a Geometric Algebra based implementation of particle dynamics. After all, in all of the above derivations, every quantity is simply a linear vector, no concept of rotation exists. Although in a linked system solved analytically, angular momentum and acceleration are obviously relevant, in the numerical methods used here they only ever exist as sources of forces on a given particle, and so there is no benefit at all to be gained from using GA in the implementation. As stated above, this section of the project was only ever intended to provide an introduction to the computational issues in a physical simulator and to build up some of the tool-set needed for the next stage of the project. At all times the possibility of a GA implementation was considered but never found to be suitable. The only case for implementing a GA based approach would be in avoiding the costs of conversion to CGA if a combined collision system for particle and solid models was implemented. This combined form of simulation is frequently employed for the simulation of cloth within clothing of computer generated film characters but has not been implemented within this project<sup>15</sup>.

<sup>15</sup>The problem of cloth simulation introduces several additional problems. An analysis of these and the solutions utilized by Pixar Studios, a leader in the field, can be found in [1]. Note, the authors both moved to Pixar shortly after publication of this paper.

## 2.3 Rigid Body Dynamics

When simulating the dynamics of rigid bodies the state vector must be extended with a further two elements. This is due to the need for the rotation of the body to be considered as well as the body's position. The exact way this is done is dependent on the way in which these quantities are stored.

### 2.3.1 Traditional Approach

The position of the body is represented as a vector from the origin to the centre of mass of the body. The rate of change of the position is dependent on the linear momentum of the body,  $P(t)$ . The rate of change of  $P(t)$  is equal to the sum of the forces applied to the body. This section is very similar to that for a particle of equivalent mass positioned at the centre of mass. Calculation of the centre of mass is explained below.

Rotation is represented via a 3x3 rotation matrix

$$R(t) = \begin{pmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{pmatrix} \quad (27)$$

In this equation the first column represents the new direction of the x axis, the second represents the new direction of the y axis and the third column represents the new direction of the z axis.

In order to simulate the motion of rigid bodies it is also necessary to find the dependence of  $\dot{R}(t)$  on the applied forces. To do this we have to introduce another element to the state, the angular momentum  $L(t)$ . Angular momentum, angular velocity and torque make use of the concept of an axial vector. With axial vectors the direction of the vector is perpendicular to the plane in which any given point in the object is 'rotating.' The axial vector's magnitude is dependent upon the rate of rotation. The rate of change of angular momentum is simply equal to the torque applied to the body.

The rate of change of the rotation matrix is dependent on the angular velocity. Angular velocity is found from the angular momentum. This dependence is given

by

$$L(t) = \int [r \times (w \times r)] dm \quad (28)$$

Now this can be rearranged to give a linear function in terms of an *Inertia Tensor*.<sup>16</sup>:

$$\begin{aligned} L(t) &= Iw(t) \\ w(t) &= I^{-1}L(t) \end{aligned} \quad (29)$$

The inertia tensor is pre-calculated for the body and stored within a data file for use if the body is imported in future scenes.

$$I_{body} = \sum \begin{pmatrix} m_i(r'_{iy}{}^2 + r'_{iz}{}^2) & -m_i r'_{ix} r'_{iy} & -m_i r'_{ix} r'_{iz} \\ -m_i r'_{iy} r'_{ix} & m_i(r'_{ix}{}^2 + r'_{iz}{}^2) & -m_i r'_{iy} r'_{iz} \\ -m_i r'_{iy} r'_{ix} & -m_i r'_{iz} r'_{iy} & m_i(r'_{ix}{}^2 + r'_{iy}{}^2) \end{pmatrix} \quad (30)$$

$I_{body}(t)$  is transformed to  $I(t)$  using:

$$I(t) = R(t)I_{body} \quad (31)$$

Now to find  $\dot{R}$  the skew symmetric matrix  $w^*(t)$  is formed from the components of the angular momentum axial vector.

$$w^*(t) = \begin{pmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{pmatrix} \quad (32)$$

Using this matrix the rate of change of the rotation matrix can be expressed as

$$\dot{R}(t) = \omega^* R \quad (33)$$

---

<sup>16</sup>Matrix Inversion is done using a Gauss Jacob algorithm with full pivoting [10].

The first order differential equation that must be solved is:

$$\frac{d}{dt} \begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ w(t)^*R(t) \\ F(t) \\ \tau(t) \end{pmatrix} \quad (34)$$

By forming all the elements on the right hand side of this equation it is possible to calculate the rate of change of the state vector elements as required for the differential solver.

It is relatively unusual to make use of rotation matrices as a way of storing data within simulations due to the susceptibility to numerical drift<sup>17</sup>. In many applications quaternions are used. This requires an added layer of complexity, either by directly manipulating the quaternions, or by converting to and from the rotation matrix form as necessary, but quaternions can be easily normalised, maintaining their pure rotation properties as required.

### 2.3.2 Geometric Algebra Approach

The fundamental difference between the conventional vector based approach described above, and one making use of Geometric Algebra is in the representation of those quantities generally represented using axial vectors<sup>18</sup>. In geometric algebra these quantities are represented by Bivectors representing a swept out area which can be more intuitively connected to the quantities represented. Looking forward to the investigation of collision detection between solid objects, the actual positions of the points defining the surfaces of the solid were stored as CGA grade 1 blades.

Initially I considered implementing the entire simulation system under the Conformal Framework. Unfortunately it is very difficult to apply standard differential solvers to this representation as the meaning of adding two elements of the algebra is not that of a shift in position. To do this it is necessary to make use of Translation Rotors and beyond the simple issue of requiring a heavily customized

<sup>17</sup>Marked effects of this numerical drift were seen whenever an object rotated rapidly. Only a few thousand time steps were needed to thoroughly distort the rotation matrix and hence the body. This affect can be seen in a Video on the Attached CD

<sup>18</sup>Axial vectors are used for representation of quantities generated via a cross product such as Torque and Angular Momentum.

solver, the overheads involved in finding these Rotors and then applying them to the objects, are great enough to outweigh the disadvantages of conversions needed if simulation is carried out in 3D Geometric Algebra and only at the end converted to the 5D representation. This also avoided the need for the calculation of a far more complex inertia tensor<sup>19</sup>.

The fundamental equation to be solved is not dissimilar to that for the Traditional approach:

$$\frac{d}{dt} \begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ -\frac{1}{2}\Omega(t)R(t) \\ F(t) \\ \tau(t) \end{pmatrix} \quad (35)$$

The torque ( $\tau$ ), the angular momentum ( $L$ ) and the angular velocity ( $\Omega$ ) are represented by bi-vectors.  $R(t)$ , the rotation of the body is a Rotor in 3D GA and so consists of 4 elements, a scalar and 3 bivectors. This representation of the rotation as 4 elements is very similar to that used with quaternions as expected and so offers considerable advantages over using the 9 elements of the rotation matrix.

As before the angular velocity must be calculated from the angular momentum. Now,

$$L(t) = \int d^3x \rho x \wedge (x \cdot \Omega(t)) \quad (36)$$

$$I_{body} = \sum \begin{pmatrix} m_i(r'_{ix}{}^2 + r'_{iy}{}^2) & m_i r'_{iy} r'_{iz} & -m_i r'_{ix} r'_{iz} \\ m_i r'_{iy} r'_{iz} & m_i(r'_{ix}{}^2 + r'_{iz}{}^2) & m_i r'_{ix} r'_{iy} \\ -m_i r'_{ix} r'_{iz} & m_i r'_{ix} r'_{iy} & m_i(r'_{iy}{}^2 + r'_{iz}{}^2) \end{pmatrix} \quad (37)$$

To understand how this  $I_{body}$  can be used, consider taking the instantaneous angular momentum of the body at its current position in space. This can then be rotated back to the fixed version of the body for which the inertia tensor has been previously calculated,  $L_{body} = \tilde{R}^L R$ . The inverse of the inertia tensor is used to convert this to  $\Omega_{body}$  which is then rotated back to the current position of the body. Thus

<sup>19</sup>In 3d GA there are only 3 bivectors whereas in CGA there are 10, hence a 10x10 tensor would be needed instead of 3x3.

we can utilise the body space inertia tensor, in place of the world space inertia tensor, to find the rate of change of rotation of the body at it's current location,  $\dot{R} = -\frac{1}{2}\Omega_{body}R(t)$ .

As with the traditional approach outlined above, numerical drift rapidly causes the rotor,  $R(t)$  to stop being a pure rotation. This can be easily overcome by recalling that  $R\tilde{R} = 1$  and rescaling all elements of R by  $\frac{1}{\sqrt{R\tilde{R}}}$ .

Now, note that the rotation rotor derived here is equally valid for points mapped to the CGA form. This is not true for the translation. To apply the translation  $x(t)$  it was necessary to find the Translation Rotor and apply this to the vertices after rotation. These two rotors need only be calculated once per body. The overhead involved in creating the rotors needed to manipulate a CGA representation of the vertices is low and is outweighed by the computational savings to be found with some forms of collision detection using CGA.

### 2.3.3 Precalculation of Solid Parameters

Both the traditional and GA based approaches to solid modelling require knowledge of the mass of the object and the inertia tensor (or strictly speaking its inverse.) These are calculated in a similar fashion for both approaches.

1. Find a bounding box around the entire object.
2. Divide bounding box up into a large number of small sections on an even grid (cells). Accuracy of estimate is controlled by size of the cells, the smaller they are, the more accurate the estimate but the longer the calculation takes.
3. Take each cell in turn and check if it lies within the solid. If it does, then add the volume of the small section being investigated to the total volume and also add  $x * dm$  to a vector defining the centre of mass (where  $x$  is the vector defining the centre of the current cell and  $dm$  is the cell mass).
4. Divide centre of mass vector by the mass to obtain the position of centre of mass.
5. Now compute the inertia tensor using similar approach to above and the knowledge of the position of the centre of mass.
6. Invert the inertia tensor using Gauss-Jacobi with full pivoting.

Two methods are implemented for evaluating if a given element is within the body. The first utilises the surface normals as explained in 2.4.3. The second method is more general as it works for any body, not just those that are convex. In this case we define a line segment from the point in question to one predefined to lie well outside the body. We then evaluate the number of surface facets crossed by the segment. If even the point is outside the body, if odd then it lies within<sup>20</sup>.

Note that, upon loading an object, the normals of every surface are calculated and stored<sup>21</sup>. By applying the current rotation to the normals, either via the rotation rotor (CGA) or rotation matrix, the surface normals' current directions are available for checking of containment or collision response evaluation (see 2.5).

## 2.4 Collision Detection

Upon completion of the basic solid body simulator described above, the next task was to detect and handle collisions between pairs of solids. Some early work had been completed on collision detection for particles and surfaces defined between sets of 3 particles. The problem with this is the indeterminate nature of collision between two point masses. To allow simple use of the various levels of collision detection explored, an operator, `||`, was defined for the class *cga\_solid*<sup>22</sup>. This operator calls a number of different functions which make use of different methods to attempt to rapidly establish if a collision has occurred. A *colclass* class is returned which contains information on the nature of the collision and provides an overloaded `==` operator to allow rapid evaluation of the type of collision. Unfortunately, the limited available time for this stage of the project restricted the extent to which the traditional methods could be implemented and so, where noted, some of the methods below were not implemented and are only included here to provide for comparisons with CGA based methods.

---

<sup>20</sup>Care is needed to avoid problems with the line segment crossing the join between two facets, so a list of current intersections detected must be checked to eliminate this case.

<sup>21</sup>In order to improve performance each normal is compared with all normals currently stored for the body and if the direction is the same as an existing normal the new one is not stored. This situation occurs whenever an object has a non-triangular surface which must be triangulated.

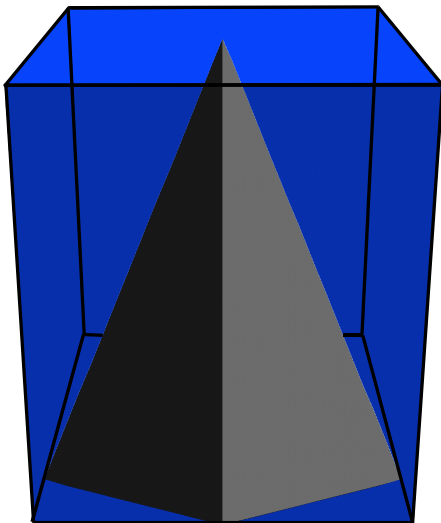
<sup>22</sup>*cga\_solid* is a large class that holds all data stored for a given body including everything from the state components, to current collision status.

### 2.4.1 Bounding Volumes

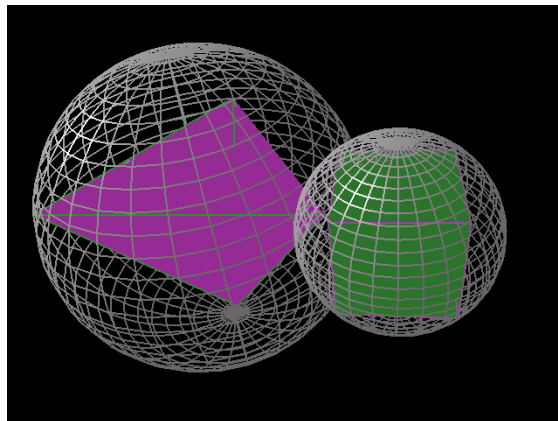
The fastest method of eliminating pairs of bodies from having collided is to discover if a pair of simpler shapes, that fully contain the solids of interest, intersect. Hence this is often used as an initial step in any collision detection algorithm. There are several obvious options for the volumes to be used.

**Bounding Cuboid** This shape is the one most used in traditional modelling systems. It is simple to form and checking for intersections between different cuboids is relatively simple. Either an axis aligned cuboid is used to minimize the complexity of intersection checks<sup>23</sup>, or an object aligned cuboid is used<sup>24</sup>. The choice made is a compromise between speed of intersection evaluation and accuracy of detection<sup>25</sup>.

Figure 5: Types of Bounding Volume



(a) Axis Aligned Bounding Cuboid



(b) Bounding Sphere

<sup>23</sup>Intersection of axis aligned cuboids can be simply achieved using 6 comparisons and fast search algorithms exist to efficiently isolate possible collisions. A simple ‘toy’ search method is described in [14].

<sup>24</sup>An Object Aligned Cuboid is the cuboid that most closely fits the object it is bounding.

<sup>25</sup>There did not appear to be any significant advantages to be gained from this technique by utilising CGA so it was not implemented and is only included here for reference.

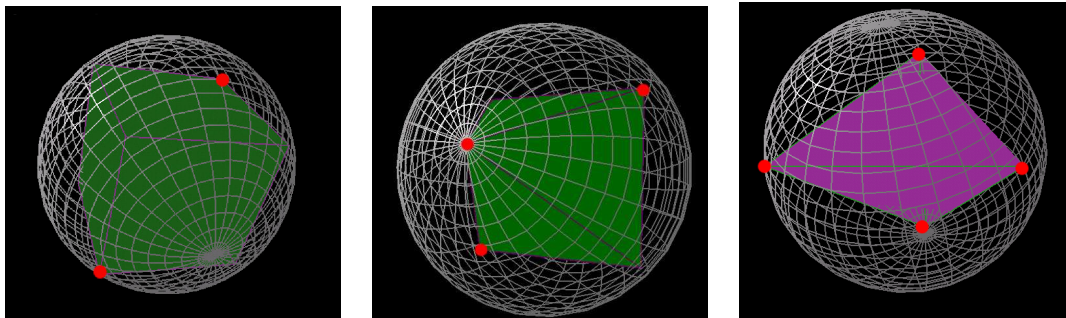
**Bounding Sphere** Spherical bounding volumes are even easier to test for intersection. However, they are much harder to form in an efficient fashion. The fastest algorithm in common use is that developed by Welzl [13].

This algorithm relies on the ability to find the smallest bounding sphere for a subset of the vertices. Thus we need to be able to form all spheres that may bound the set of vertices being considered. The trivial case of one point is obviously bounded by a sphere of zero radius. Two points must be bounded by a sphere centred mid way between them. Unfortunately for larger numbers of points the sphere may be controlled by anything from 2 to 4 points from within the set of those to be contained. Thus, 3 classes of sphere may need to be evaluated.

Considering an arbitrary solid body, the smallest possible bounding spheres are those controlled by only two vertices of the object as in Fig. 7(a). The centre of the sphere is halfway between the two points and the radius is half the distance between them. The sphere is then formed using the dual form  $M^*$  which is shown in [7] to be  $M^* = C - \frac{1}{2}\rho^2n$  where  $C$  is the centre point and  $\rho$  the radius.

If we assume that a traditional implementation will represent the sphere as a centre point and radius then there is a small overhead in storage requirements with the CGA representation of a sphere having 5 degrees of freedom to the 4 for the traditional representation.

Figure 6: Formation of Bounding Spheres - the 3 cases



(a) 2 Point Sphere(Cube)

(b) 3 Point Sphere (Pyramid)

(c) 4 Point Sphere (Tetrahedron)

If all points of the body are contained within a sphere defined by 2 points then this is the smallest possible bounding sphere and there is no need to consider the more complex cases.

If not, then spheres defined by 3 points, e.g. Fig. 7(b), must be considered. In this case the sphere is formed by first forming a circle through the 3 points. From this circle the centre point and radius are extracted and then the sphere is formed as above. The extraction of the centre point of a circle in traditional vector geometry is a non trivial problem and so CGA offers considerable advantages here. To find the centre of a circle defined by 3 points we first transform the points into a new coordinate system aligned with the plane of the points.  $(x_1, y_1)$ ,  $(x_2, y_2)$ ,  $(x_3, y_3)$  the following equations are used.

$$x = \frac{1}{2}N_1D^{-1}, \quad y = \frac{1}{2}N_2D^{-1} \quad (38)$$

where,

$$N_1 = \begin{vmatrix} (x_2^2 + y_2^2 - x_1^2 - y_1^2) & (y_2 - y_1) \\ (x_3^2 + y_3^2 - x_1^2 - y_1^2) & (y_3 - y_1) \end{vmatrix}$$

$$N_2 = \begin{vmatrix} (x_2 - x_1) & (x_2^2 + y_2^2 - x_1^2 - y_1^2) \\ (x_3 - x_1) & (x_3^2 + y_3^2 - x_1^2 - y_1^2) \end{vmatrix}$$

$$D = \begin{vmatrix} (x_2 - x_1) & (y_2 - y_1) \\ (x_3 - x_1) & (y_3 - y_1) \end{vmatrix}$$

In CGA the location of the centre is simply  $C = MnM$  where M is 3-blade defining the circle. Clearly in both cases, finding the radius is simple once the centre is known.

If all the possible 3 point defined spheres still fail to contain all the vertices of the body, then the smallest bounding sphere must be controlled by 4 points. In CGA to form this sphere is simply a matter of forming a 4-blade directly from the points. Using traditional methods there is an equivalent method to that given for circles above. This method involves the inversion of a 3x3 matrix, a significant computational task when applied many times to consider all possible sets of 4 points in a complex object.

Welzl's Algorithm is an iterative method based on Seidel's Linear Programming Algorithm with the addition of a heuristic for increased speed. The heuristic is based on the observation that points furthest from a previously checked sphere are likely to lie outside any new sphere formed, and thus a sort process ensures these are

checked first for each new sphere. Yet again the key to this geometric algorithm is to eliminate as many false cases as fast as possible. Although implementing Welzl's Algorithm would have been interesting, there do not appear to be any significant benefits to be had from the use of CGA beyond those inherent in the formation of the various classes of sphere described above. As a result, the approach used is an exhaustive search, which is far less efficient. All spheres in a given class are evaluated and the smallest bounding sphere is selected<sup>26</sup>.

Further approaches to bounding volumes and their use are described [6].

With a CGA based sphere definition, intersections between spheres, are evaluated by finding the Meet (see 1.6) of the two spheres, here denoted  $C$ . An intersection exists if  $C^2 \geq 0$ . The first collision response model implemented here was based upon the impacts occurring between bounding spheres instead of the actual bodies. This technique is frequently used in computer games, where speed of response is far more important than accuracy. The restrictions on the depth of intersection were based upon evaluating the radius and applying a maximum acceptable value. The collision response was then evaluated as with the more complex cases considered below.

Several issues had to be addressed with the definition of a sphere as the outer product of 4 surface points. The first is concerned with evaluating if a point lies inside the sphere or outside the sphere. This can be easily evaluated by taking the dot product of the point with  $M^*$ , the 1-blade formed by taking the dual of the 4-blade form. Unfortunately there is no convention for the sign of the terms in this form as they are dependent on the order in which the points are used in finding the 4-blade. The sign of the terms effectively dictates the direction of the normals. What is needed is a convention similar to the anticlockwise convention for planar surfaces, but this is easier said than done with 4 points. Thus the approach adopted here is to define points inside as having a dot product with the sphere of less than 0. The point at infinity,  $n$ , can then be checked and all component signs inverted if necessary to comply with this convention.

---

<sup>26</sup>Due to the inherent complexity of a traditional implementation, only a CGA based approach was tested.

### 2.4.2 Convex Solid

Due to the extensive nature of the subject of collision detection, a full general implementation was not practical within this project. As a result the subset of convex bodies has been chosen for investigation. Most of the general techniques described here are also applicable to concave bodies but the actual geometric algorithms are considerably more complex<sup>27</sup>.

Fig. 7 shows the way that various techniques are combined to discover whether a collision has occurred and the nature of the collision. Not all possible collision types are considered and distinguishing between various types of collision can prove extremely difficult. For more in depth evaluation of these issues, see [14].

### 2.4.3 Point in Body

This algorithm establishes if any vertex of one body lies within the other. Any that are, prove a collision has occurred.<sup>28</sup> There are a number of different approaches, dependent on the restrictions on the types of solids being simulated. In the simple case of convex solids that is considered here, the fastest method is to take the dot product of a vector defined from a point on each facet to the point in question, with the normal of that facet. If any of these dot products are positive then the point must lie outside the object.

This dot product gives a measure of perpendicular distance of the point from the surface. If a tolerance is defined for the distance a point may enter into a body; then values greater than the tolerance will indicate that the time bisection approach should be used to ‘retreat’ to a point where an acceptable level of penetration has occurred. If the magnitude is less than the tolerance then we have a collision to respond to<sup>29</sup>.

To process a collision we need to know the point of impact and a normal vector between the two colliding bodies. In the case of a point surface impact this is given

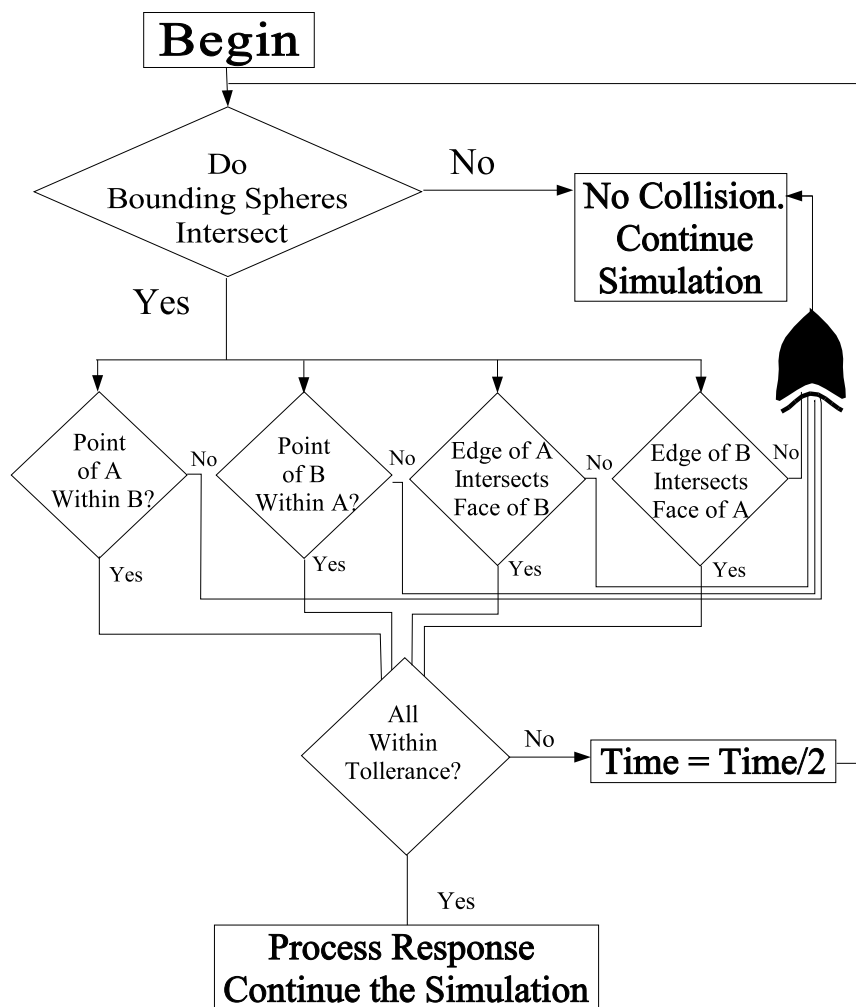
---

<sup>27</sup>Rapid collision detection for non-convex models is a major area of ongoing research within this field.

<sup>28</sup>The software assumes that there are no collisions on initialization of collision detection. If this is not done then it is necessary to handle the indeterminate case of two stationary intersecting bodies.

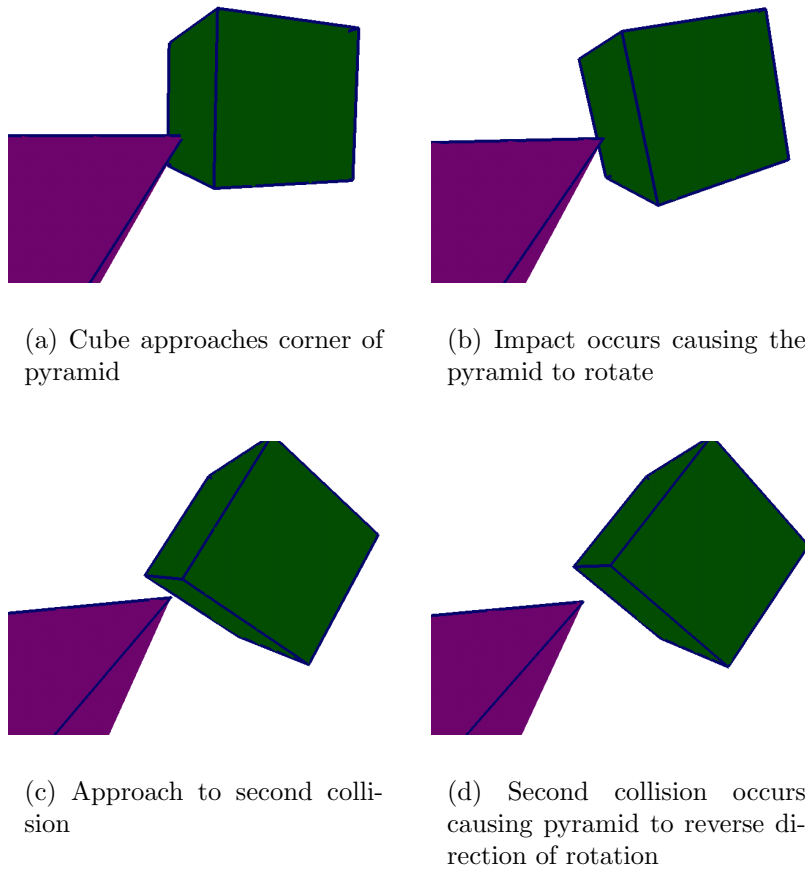
<sup>29</sup>Note that we must ensure that all collisions currently occurring meet the tolerance requirements before processing any responses. If not, then we must move backwards through time to a point at which they do.

Figure 7: Implemented Collision Detector



approximately by the position of the point and the surface normal of the surface penetrated. Fig. 8 shows the results when a lateral force on a cube pushes it into a point of a stationary pyramid.

Figure 8: Response to a Point Contact



#### 2.4.4 Segment Triangle Intersection

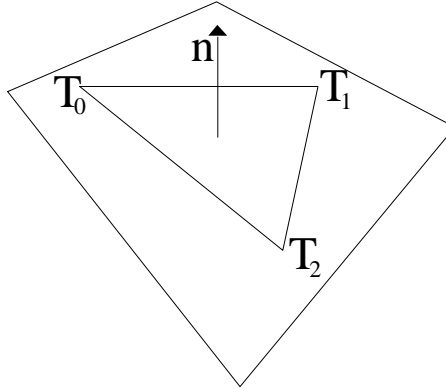
As shown in Fig. 7 there is another type of collision detector required to establish if a collision has occurred. Any intersections between edges of facets and surfaces that are not due to a vertex entering the other body must also be discovered. The conventional methods for investigating this and those using CGA are explained below. Note, simple Geometric Algebra provides no benefits at all in this case.

**Non-GA** There are many algorithms which can be used to discover if such an intersection exists using only vector based methods. One of the most computationally efficient is the parametric approach outlined here [11]. The algorithm is a two stage process.

1. Find the intersection of the line on which the segment lies with the plane on which the triangle lies. (may not exist in which case the segment clearly does not intersect the triangle)
2. Discover if this point lies within the triangle defining the surface

The first stage relies on describing the line in parametric form,  $P(r) = \mathbf{L}_0 + r(\mathbf{L}_1 - \mathbf{L}_0)$  and the plane by a point  $V_0$  and normal vector  $\mathbf{n}$  (Fig. 9). The vector  $\mathbf{n}$  is the cross product of two triangle edges:  $\mathbf{n} = (\mathbf{T}_0 - \mathbf{T}_1) \times (\mathbf{T}_0 - \mathbf{T}_2)$ . Having

Figure 9: Definition of Plane



found  $\mathbf{n}$  the point of intersection can be rapidly found using:

$$r_I = \frac{\mathbf{n} \cdot (\mathbf{T}_0 - \mathbf{L}_0)}{\mathbf{n} \cdot (\mathbf{L}_1 - \mathbf{L}_0)} \quad (39)$$

If  $r_I \geq 0$  &  $r_I \leq 1$  then the intersection is on the segment. If not there is no intersection.

Now that the segment / plane intersection is known, the final check is to discover if it lies within the triangle on the plane. This is done by transforming the point of intersection into a parametric form in terms of two of the sides of the triangle.

The parametric representation of the plane is

$$P(s, t) = \mathbf{T}_0 + s(\mathbf{T}_1 - \mathbf{T}_0) + t(\mathbf{T}_2 - \mathbf{T}_0) = \mathbf{T}_0 + s\mathbf{u} + t\mathbf{v} \quad (40)$$

If the point of intersection,  $\mathbf{I}$ , is within the triangle and we convert its coordinates into the parametric form defined by the triangle, then  $s \geq 0$ ,  $t \geq 0$  and  $s + t \leq 1$ . To convert to this form we define  $\mathbf{w} = \mathbf{I} - \mathbf{T}_0$ . Thus we want to solve the equation  $\mathbf{w} = s\mathbf{u} + t\mathbf{v}$ . To do this, vectors in the plane are found that are perpendicular to each of the side vectors and of the same length as them. Thus,  $\mathbf{v}^\perp = \mathbf{n} \times \mathbf{v}$  and  $\mathbf{u}^\perp = \mathbf{n} \times \mathbf{u}$ . This is then used to derive equations 41 and 42.

$$\begin{aligned} \mathbf{w} &= s_I \mathbf{u} + t_I \mathbf{v} \\ \mathbf{w} \cdot \mathbf{v}^\perp &= s_I \mathbf{u} \cdot \mathbf{v}^\perp + t_I \mathbf{v} \cdot \mathbf{v}^\perp \\ s_I &= \frac{\mathbf{w} \cdot \mathbf{v}^\perp}{\mathbf{u} \cdot \mathbf{v}^\perp} = \frac{\mathbf{w} \cdot (\mathbf{n} \times \mathbf{v})}{\mathbf{u} \cdot (\mathbf{n} \times \mathbf{v})} \end{aligned} \quad (41)$$

$$t_I = \frac{\mathbf{w} \cdot \mathbf{u}^\perp}{\mathbf{v} \cdot \mathbf{u}^\perp} = \frac{\mathbf{w} \cdot (\mathbf{n} \times \mathbf{u})}{\mathbf{v} \cdot (\mathbf{n} \times \mathbf{u})} \quad (42)$$

**CGA based approach** This approach is based upon that given in [7].

Again the problem is split into the two stages we had before, first we find the intersection between the ray defined by the segment and the plane defined by the triangle. To do this we make use of the meet operator ( $\vee$ ) as defined in 1.6. In the case of a line / plane intersection the meet is given by a bivector of the form  $B = X \wedge n$ . Thus we need to extract the actual point of intersection. This can be done by simply equating the components of the vector  $\mathbf{x}$ ,  $x_i$  to those of the  $e_i \wedge n$  terms.

As with the traditional approach we now need to discover if this point of intersection lies on the segment and within the triangle. To evaluate if the point of intersection lies on the segment, a series of comparisons are used to ensure it lies within the bounding box defined by the two ends of the segment.

Finally we evaluate if the point lies within the defined triangle. The plane is defined by the 4 blade  $\Phi = P_1 \wedge P_2 \wedge P_3 \wedge n$  (see Tab. 1). To do this we define a

reciprocal frame,  $P^1, P^2, P^3$  satisfying

$$P^i \cdot P_j = \delta_j^i \text{ for } i, j = 1, 2, 3 \quad (43)$$

where

$$P^i = |\Phi|^{-2}(n \wedge L_i) \cdot \Phi \quad (44)$$

and  $L_1 = P_2 \wedge P_3, L_2 = P_3 \wedge P_1, L_3 = P_1 \wedge P_2$  and  $|\Phi| = \sqrt{-\Phi^2}$ . Now if we have a point of intersection  $I$  then it can be represented by

$$I = \lambda_1 P_1 + \lambda_2 P_2 + \lambda_3 P_3 + \lambda_n n \quad (45)$$

The  $\lambda$ 's can then be recovered using the reciprocal vectors defined above.

$$\lambda_i(P^i \cdot P) \text{ for } i = 1, 2, 3 \quad (46)$$

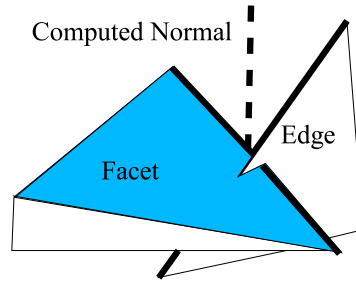
If  $0 \leq \lambda_i \leq 1$  for  $i = 1, 2, 3$  then the intersection is within the triangular facet.

If an edge / facet intersection has occurred, the common normal of the two bodies is needed to compute the collision response. It is crucial to note that although the intersection is detected by considering edge / facet crossings, the intersection itself will have been an edge passing through an edge. It is the common normal of these two edges that is required. The nearest facet edge to the point of intersection is located and is assumed to be the one crossed<sup>30</sup>. This is found by reflecting the point in the line and finding the square of the difference in coordinates. The smallest result to this calculation indicates the nearest side to the point. When the two edges in question have been isolated, the normal can be found using a conventional cross product.

---

<sup>30</sup>This assumption is not always correct, particularly when the dimensions of the face are of similar order to the tolerance allowed in the intersection depth.

Figure 10: The normal for an edge / facet intersection



## 2.5 Collision Response

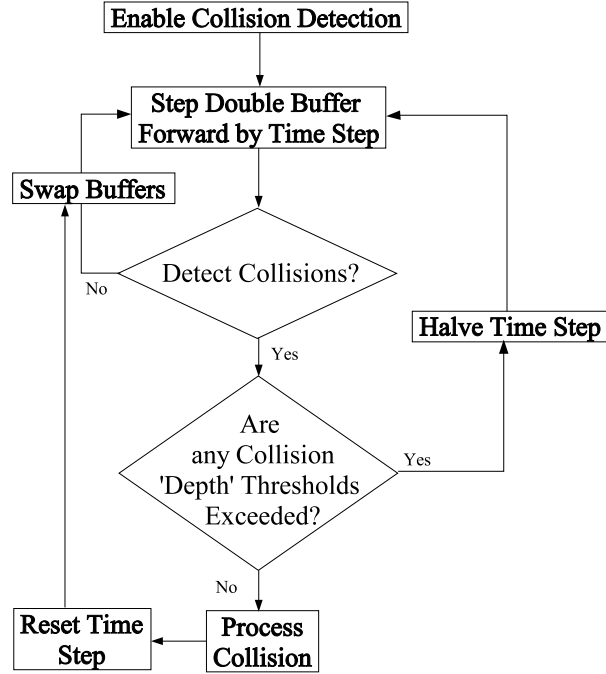
### 2.5.1 Time Bisection and Double Buffering

Before a collision response can be computed it is necessary to start from a point in time at which the collision has just occurred. The acceptable level of collision is controlled by various thresholds based on depth of intersection. The values are set to ensure accuracy to a visually undetectable level. In order to reach this point, a simple bisection<sup>31</sup> based approach is used as shown in Fig. 2.4.4. To allow the stepping backwards in time required, it was necessary to add a level of abstraction to the storage of objects within the program. This abstraction was a form of double buffering, similar to that used for recent computer graphics. By using this technique, a copy of the last ‘good’ data set can always be maintained, with the results of the step stored in the alternative buffer. On acceptance of the step having met collision requirements (either none occurred or responses computed), the buffers are swapped, leading to an update in the displayed data and the continuation of the simulation.

The response of a body to any collision detected is evaluated between steps of the differential solver. This is because a collision between non-deformable bodies results in an impulse and this is a non linear event. Thus this magnitude and direction of the impulse are evaluated and the resulting changes to linear and angular momentum computed and applied to the object’s stored state.

<sup>31</sup>Alternative possible methods are discussed in 4.3.2.

Figure 11: The Program Flow when Collision Detection is Enabled



### 2.5.2 Traditional Approach

The rate of change of position of the point of contact (on the body A) immediately following the impact is given by:

$$\dot{p}_a^+ = v_a^+ + \omega_a^+ \times r_a \quad (47)$$

now if we define the impulse as  $J = j\hat{n}$  then,

$$v_a^+ = v_a^- + \frac{j\hat{n}}{M_a} \quad (48)$$

$$\omega_a^+ = \omega_a^- + I_a^{-1}(r_a \times j\hat{n}) \quad (49)$$

combining (47) and (48):

$$\dot{p}_a^+ = \dot{p}_a^- + j \left( \frac{\hat{n}}{M_a} + I_a^{-1}(r_a \times \hat{n}) \right) \times r_a \quad (50)$$

similarly (noting opposite direction of impulses on B,

$$\dot{p}_b^+ = \dot{p}_b^- - j \left( \frac{\hat{n}}{M_b} + I_b^{-1}(r_b \times \hat{n}) \right) \times r_b \quad (51)$$

Thus the parting velocity is

$$\begin{aligned} \dot{p}_a^+ - \dot{p}_b^+ &= (\dot{p}_a^- + \dot{p}_b^-) \\ + j \left( \frac{\hat{n}}{M_a} + \frac{\hat{n}}{M_b} + (I_a^{-1}(r_a \times \hat{n})) \times r_a + (I_b^{-1}(r_b \times \hat{n})) \times r_b \right) \end{aligned} \quad (52)$$

Now,  $v_{rel}^+ = \hat{n} \cdot (\dot{p}_a^+ - \dot{p}_b^+)$  and  $v_{rel}^+ = \epsilon v_{rel}^-$ , where  $\epsilon$  is the coefficient of restitution. So rearranging the magnitude of the impulse is

$$j = \frac{-(1 + \epsilon)v_{rel}^-}{\frac{1}{M_a} + \frac{1}{M_b} + \hat{n} \cdot (I_a^{-1}(r_a \times \hat{n})) \times r_a + \hat{n} \cdot (I_b^{-1}(r_b \times \hat{n})) \times r_b} \quad (53)$$

Thus by solving this relatively simple equation we can find the impulse and using (47) we can establish the required changes to body linear and angular velocity.

### 2.5.3 Geometric Algebra Approach

As with all the actual mechanics within this project, there is no real advantage to be had in working with CGA entities. Again we would need to have addition of vector elements readily available.

The derivation of the geometric algebra approach very closely follows that outlined for traditional algebra above.

$$j = \frac{-(1 + \epsilon)v_{rel}}{\frac{1}{M_a} + \frac{1}{M_b} + \hat{n} \cdot (r_a \cdot I_a^{-1}(\hat{n} \wedge r_a)) + \hat{n} \cdot (r_b \cdot I_b^{-1}(\hat{n} \wedge r_b))} \quad (54)$$

## 2.6 CGA / GA versus Traditional

The derivations and discussions above indicate that there is very little to choose between GA and more traditional methods when it comes to the simulation of motion without inter-body interaction. The decision on which to use for a given problem is largely based upon the experience of the programmer with the two different mathematical approaches. Whilst GA is in many ways more intuitive to

use, most practicing programmers are liable to have greater experience with vector algebra based methods.

It is in the area of collision detection that the advantages of CGA really come to the fore. As shown above, the ease of manipulating spheres and planes leads to considerable simplification of some of the mathematics involved in detection of collisions. This advantage is particularly pronounced in the problem of forming smallest bounding spheres for a given set of vertices.

Examples of the various types of collision can be seen in video files on the attached CD.

### 3 Conclusions

*“Whether the use of Geometric Algebra provides any advantages, in terms of coding simplicity or speed”*, is a somewhat hard question to answer based upon the depth of investigation possible within the time constraints of this project. It is also a gross simplification of the issue, as the field of physical modelling is extremely varied and encompasses an enormous variety of problems and solutions. Hopefully, this report has given the reader an introduction to a few of these areas.

The only advantages seen with the use of basic GA lie in the simplicity of understanding and hence coding. However, this is very dependent on the experience of the programmer with the relevant mathematical techniques. In the latter stages of the project, the experience of GA gained, meant that its use became as intuitive as traditional vector based approaches. Alongside this the author developed an appreciation of the elegance and clarity that GA can provide. It should be noted that, until these latter stages, experience in the use of GA was insufficient and the initial approach adopted to a given problem was to code up a traditional algorithm and then attempt to convert it - an approach that possibly limited the techniques developed for the use of GA.

The story for CGA was somewhat different. It is interesting to note that all the use of CGA was concerned with the manipulation of the geometry rather than directly within the simulation of the mechanics. In almost every single case it proved easier to conceive of solutions to problems using CGA based techniques than traditional methods<sup>32</sup>. More use of CGA would have been made as a storage

---

<sup>32</sup>Only with ray facet intersections, was the traditional approach found to be slightly easier to

method so as to avoid the need for conversion, except that the concept of an incremental change required by the differential solver does not exist.

Bar a few specific cases, it is likely that, in comparison with traditional methods, there will always be a computational overhead for GA/CGA based algorithms. This overhead is partly caused by the years of refinement and optimization that the conventional methods have undergone. This advantage of traditional methods should be eroded over time as these new (GA/CGA) algorithms and methods develop.

## 4 Future Work

The limits on time available for the project led to a number of less than ideal compromises. Several areas of possible interest were not investigated fully. Some of these look to be promising areas for the use of CGA in particular (See Sphere Trees, 4.2.2). Also, in the latter stages of the project, a full investigation of traditional methods became impractical. Although a full traditional implementations of everything were not strictly necessary to evaluate the potential of GA/CGA approaches, the highly developed existing techniques often provide insights into more advanced GA/CGA algorithms. Conversely, as previously mentioned, the use of a traditional implementation may have limited the directions of investigation when formulating the GA/CGA alternative.

### 4.1 Hardware Implementation

The inherently parallel nature of many of the operations, such as the 3 basic products, may mean the development of dedicated hardware will be a factor in overcoming the issue of speed, conceivably making a geometric product, between arbitrary multivectors, as fast as a traditional float\*float operation.

As seen throughout the above work, the advantages of CGA are mainly found within the handling of the geometry of the system rather than with the actual mechanics, e.g. useful for discovering whether collisions have occurred, but not for evaluating the results of these collisions. Thus, it seems probable that the areas in

---

understand, but given the similarity of the two methods, this may again be due to the author's lack of experience with these techniques.

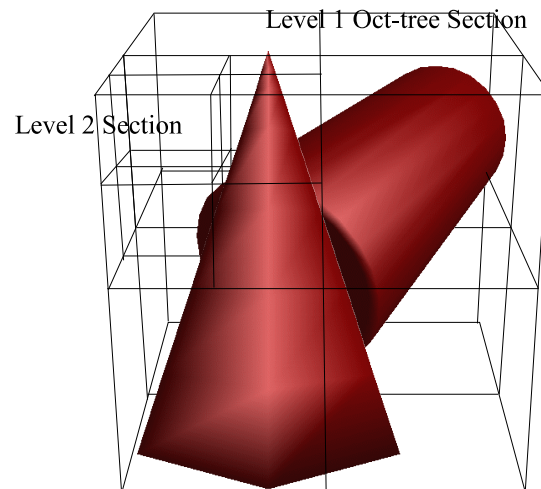
which advantages from hardware acceleration are most likely to be seen are those which involve a lot of manipulation of geometry.

## 4.2 Advanced Collision Detection

In the collision detection approach adopted within this project, the first stage is to eliminate as many prospective collision pairs as possible using a bounding volume, be it a sphere or a cube. The advantage of doing this lies in the much simpler collision tests for such a bounding volume in comparison with the actual bodies. This approach can be extended to eliminate further possible collisions through the use of multiple simple geometric shapes.

### 4.2.1 Oct-trees

Figure 12: Oct-tree based Collision Detection

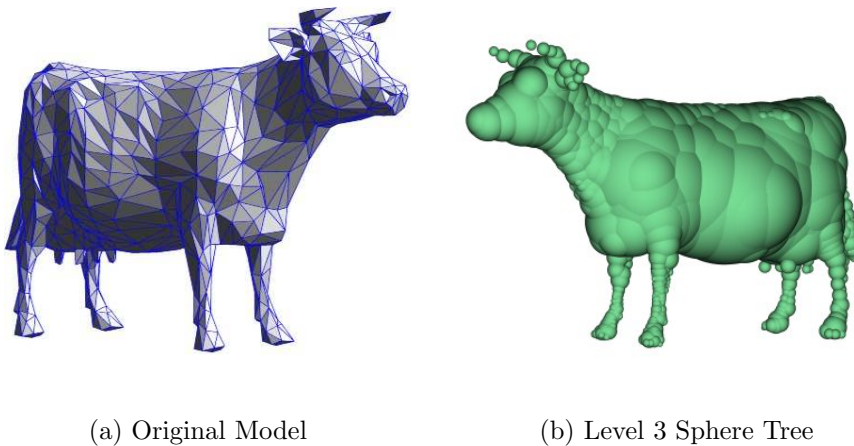


The simplest approach is an extension of the bounding cuboid. Eight new cuboids are defined by splitting the original cuboid into 8 identical ‘child’ cuboids. The second stage is to evaluate which of these new cuboids actually contain part of the body being approximated. This process can then be continued to produce an increasingly accurate estimate of the body, sometimes termed the packing crate estimate. This entire process can be preprocessed as the form of the tree does not change and can simply be rotated and translated as necessary to follow the body.

When the possibility of collision is being evaluated there exist fast search methods designed to rapidly eliminate possible collisions. The ease of manipulation of planes may make this algorithm easier to implement in CGA than using traditional methods but greater advantages seem likely to be found with the use of a sphere based structure.

#### 4.2.2 Sphere-trees

Figure 13: Sphere Tree Based Models for use in Collision Detection



The formation of sphere-trees is considerably more complex than the formation of oct-trees described above. However, the principle of building up an increasingly accurate approximation to the body and the formation of a searchable tree remains. An example of the results that can be obtained from advanced algorithms can be seen in Fig. 13. The algorithm used to create these examples is that advocated by G. Bradshaw and C. O’Sullivan [2] and utilises an iterative method based upon an extension of Welzl’s algorithm utilising information derived from the Vornoi Diagram<sup>33</sup>. The construction of a Vornoi Diagram involves the creation of spheres through multiple points and as such this immediately appears likely to benefit from the use of CGA. In order to provide a tree with a unique path to any point,

---

<sup>33</sup>Specifically an approximation to the Medial Axis. The Medial Axis is the surface that separates regions of space in which the nearest body is always the same. The approximation works by defining the bodies as an increasingly large set of points, instead of the harder problem of handling the surfaces directly.

shapes formed by removing part of a sphere on one side of a plane are used. The unified nature of spheres and planes in CGA should allow such compound bodies to be easily manipulated, providing further advantages. If a sufficiently accurate approximation is obtained then the collision responses can be calculated merely by using the sphere tree itself rather than the actual body - this avoids several complicated cases as only point impacts are possible between spheres whilst planar bodies allow edge to plane impacts and plane to plane impacts.

### 4.3 Finding Time of Collision

The bisection method was selected so as to minimize the changes needed to the basic differential solver, but it is a far from efficient method. Bisection does not provide particularly good convergence. This can be addressed by better search techniques. It also requires the entire systems state to be evaluated a large number of times if high accuracy is required. It is possible that Interpolation of the Rotors may avoid this need (see 4.3.2).

#### 4.3.1 More Advanced Optimization Methods

This is a well understood area and so won't be considered in depth here. A method such as a Golden Section line search based on minimizing the distance between surfaces should produce fairly rapid convergence without the increase in complexity a gradient based method would require. This would come at the cost of greater storage requirements.

#### 4.3.2 Interpolation of Rotors

Recent work by Richard Wareham [9] has shown that general Rotors in CGA can be interpolated reasonably simply. Although time has not allowed full consideration or testing of the consequences of such interpolation, it does provide a possible alternative to recalculating the entire system motion many times.

# Appendix

## A Programming Methodology

In order to allow use of the  $\mathbb{R}^3$  form of Geometric Algebra (GA) alongside the Conformal Model it was necessary to produce a library able to provide all of the necessary mathematical operations. This library was written in C++, unlike *LibGA* on which it is based <sup>34</sup> [12] and so takes full advantage of C++'s simpler memory handling and, crucially from the point of view of later code, operator overloading. Unfortunately the operators that can be overloaded are somewhat restricted and so symbols commonly employed in GA notation could not be used. Instead the symbols given in *Table 3* were used<sup>35</sup>.

The main CGA element is a general multivector addressed through the use of a smart pointer (`cga_multiv_t`)<sup>36</sup> The use of smart pointers allows greater automation of memory allocation leading to a more maintainable library and generally stabler programs. The operator functions called make full use of *Grade Tracking* as developed by Richard Wareham. In brief, this entails monitoring the existence of different blades within the multivector to ensure that no unnecessary products are calculated between non-existent elements. For a full explanation see [12].

Table 3: Symbols Used for standard operations inside LibGA

| Operation         | Symbol |
|-------------------|--------|
| Geometric Product | *      |
| Outer Product     | %      |
| Inner Product     | ,      |

During the course of the project it became apparent that, in many cases, considerably more memory was allocated for multivectors than required. In many of the algorithms above, only the elements defining one grade of the multivector are

---

<sup>34</sup>LibGA is a library created by Richard Wareham for the manipulation of several different forms of Geometric Algebra and uses C.

<sup>35</sup>With hindsight the use of , was unwise as it requires more care to ensure misinterpretation by the compiler does not occur.

<sup>36</sup>Equivalent GA multivector is `ga_multiv_t`.

non-zero. As it is desirable for a physical system simulation to be able to handle very large numbers of entities unnecessary memory allocation<sup>37</sup> should be minimized. To overcome this issue, single grade elements `_1.t` (a 1-blade) and `_2.t` (a 2-blade) were introduced with all required operators defined<sup>38</sup>. A full evaluation of the advantages of implementing these elements would require a statistical analysis of the different forms of multivector that occur. What is clear is that the memory savings should be considerable. For example a 1-blade has only 5 components, in comparison with the 32 required for a full multivector. The use of these single grade elements also resulted in increased clarity of coding as the grade of element produced by each call to the product operators would be fixed.

Operator overloading makes it possible to write equations in approximately the form used when working them out by hand, or when using a package such as *Maple*. These ‘library’ functions were checked for errors by manipulating simple results and comparing them with the output of *Maple* using a suitable GA environment.

Fig. 14 gives a typical function taking advantage of all of the methods described above. Its purpose is to construct a valid sphere multivector (in this case the 1-blade version). As can be seen, it is still useful to be able to directly access the multivector elements when a specific element needs to be adjusted. Obviously this could be implemented as a function, but as it only appears relevant for flipping the normals of the spherical shell, there is no real point.

## B Differential Equation Solvers

One of the library of ‘tools’ built up early in the project was a fast differential equation solver. The range of common solvers available and the lack of clear information on their relative advantages in these circumstances meant that the only sure way of selecting the most suitable solver lay in implementing the most common types and evaluating their performance. Clearly their overall performance is very dependent on the exact nature of the problem. As a result of this the code was

---

<sup>37</sup>Large memory usage is both a problem in terms of maximum memory available and more importantly (in these days of low cost memory) it can result in long waits whilst the processor fetches data from memory.

<sup>38</sup>Although it would be beneficial to similarly allow grade 3 - 5 blades, these are more rarely present within the code and sufficient time was not available to implement and test the required operators.

Figure 14: An example function illustrating the available products and direct element access

```

cga_multiv_t make4pointsphere(_1_t& one, _1_t& two, _1_t& three, _1_t& four)
{
  \\Build the 4-blade representing the spherical shell
  cga_multiv_t sphere = one%two%three%four;
  \\Check direction of normals and flip if necessary
  cga_multiv_t check = ((cga_std.n),(sphere));
  if(check->components[0] > 0)
  {
    sphere->components[30] = -sphere->components[30];
    sphere->components[29] = -sphere->components[29];
    sphere->components[28] = -sphere->components[28];
    sphere->components[27] = -sphere->components[27];
    sphere->components[26] = -sphere->components[26];
  }
  \\Convert to the dual form for later manipulation
  sphere = cga_std.pseudo*sphere;
  return sphere;
}

```

structured to allow all these implementations to be rapidly selected for evaluation under a new set of circumstances. All these algorithms were adapted from those explained in *Numerical Recipes in C++* [10].

## B.1 1st Order Euler Method

This particular method provides a far from optimal solution due to its low accuracy and efficiency. The reason for its implementation is that, as the simplest solver, it provides a good basis for testing other sections of the code due to its predictability. Equation (55) gives the basic form.  $f(x, y)$  is the gradient evaluated at point  $y$  at time  $x$  and  $h$  is the time step.

$$y_{n+1} = y_n + hf(x_n, y_n) \quad (55)$$

## B.2 Runge-Kutta Methods

### B.2.1 4th Order Fixed Time Step

An extension of the basic *Euler* method described above that makes use of a symmetric approach to ensure that the order of the error is reduced to 5. Thus we are able to take much longer steps. In many cases the advantage gained from these longer steps outweighs the extra calculation required for each individual step. The approximation is calculated using

$$\begin{aligned}
 k_1 &= hf(x_n, y_n) \\
 k_2 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \\
 k_3 &= hf\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \\
 k_4 &= hf(x_n + h, y_n + k_3) \\
 y_{n+1} &= y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5)
 \end{aligned} \tag{56}$$

## B.3 Adapting the Time Step

Given the variation in possible setups, and in the properties of a given setup over time, it is impossible to set the time step used at a value which will prove ideal for all situations. The alternative is to use a measure of the error in each step to control the time steps taken.

### B.3.1 Double Step Approach for Adaptive Time Steps

The simplest approach is to estimate the error based upon the principle that the error is less for smaller steps. Therefore by taking two half steps and comparing the result with taking the step in one go, an estimate of the 5th order error (for a 4th order method) can be made and this used to adjust the step size.

### B.3.2 5th Order Runge Kutta with Adaptive Step-Size Using Embedded 4th Order

The general approach used for this solver is very similar to that used for the 4th order Runge-Kutta algorithm described above. A particular 5th order *Runge-Kutta* algorithm using *Cash - Karp* parameters (Table 4) has the special property that a

different combination of the components gives a 4th order accurate result alongside the 5th order version. The difference between the calculated results for the two different order solvers allows the accuracy of the 4th order solver to be estimated.

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf(x_n + a_2h, y_n + b_{21}k_1)$$

$$k_3 = hf(x_n + a_3h, y_n + b_{31}k_1 + b_{32}k_2)$$

$$k_4 = hf(x_n + a_4h, y_n + b_{41}k_1 + b_{42}k_2 + b_{43}k_3)$$

$$k_5 = hf(x_n + a_5h, y_n + b_{51}k_1 + b_{52}k_2 + b_{53}k_3 + b_{54}k_4)$$

$$k_6 = hf(x_n + a_6h, y_n + b_{61}k_1 + b_{62}k_2 + b_{63}k_3 + b_{64}k_4 + b_{65}k_5)$$

$$y_{n+1} = y_n + c_1k_1 + c_2k_2 + c_3k_3 + c_4k_4 + c_5k_5 + c_6k_6 + O(h^6) \quad (57)$$

$$y_{n+1}^* = y_n + c_1^*k_1 + c_2^*k_2 + c_3^*k_3 + c_4^*k_4 + c_5^*k_5 + c_6^*k_6 + O(h^6) \quad (58)$$

Table 4: Cash-Karp Parameters for Embedded Runge-Kutta Method

| i   | $a_i$          | $b_{ij}$             |                   |                     |                        |                    | $c_i$                 | $c_i^*$ |
|-----|----------------|----------------------|-------------------|---------------------|------------------------|--------------------|-----------------------|---------|
| 1   |                |                      |                   |                     |                        | $\frac{37}{378}$   | $\frac{2825}{27648}$  |         |
| 2   | $\frac{1}{5}$  | $\frac{1}{5}$        |                   |                     |                        | 0                  | 0                     |         |
| 3   | $\frac{3}{10}$ | $\frac{3}{40}$       | $\frac{9}{40}$    |                     |                        | $\frac{250}{621}$  | $\frac{18575}{48384}$ |         |
| 4   | $\frac{3}{5}$  | $-\frac{9}{10}$      | $\frac{6}{5}$     |                     |                        | $\frac{125}{594}$  | $\frac{13525}{55296}$ |         |
| 5   | 1              | $-\frac{11}{54}$     | $\frac{5}{2}$     | $-\frac{70}{27}$    | $\frac{35}{27}$        | 0                  | $\frac{277}{14336}$   |         |
| 6   | $\frac{7}{8}$  | $\frac{1631}{55296}$ | $\frac{175}{512}$ | $\frac{575}{13824}$ | $\frac{44275}{110592}$ | $\frac{253}{4096}$ | $\frac{512}{1771}$    |         |
| j = |                | 1                    | 2                 | 3                   | 4                      | 5                  |                       |         |

### B.3.3 Variation of Time Step

Using the above methods it is possible to produce an estimate of the 5th order error for a given step size. This can then be used to adjust the step size as given by

$$h_0 = h_1 \left| \frac{\Delta_0}{\Delta_1} \right|^{0.2} \quad (59)$$

Note that when using this equation,  $\Delta_0$ , the desired accuracy is a vector containing the allowable errors for all of the linked differential equations. It is therefore crucial that only the highest error (relative to allowable error) is considered as this alone dictates the required time step<sup>39</sup>. Ideally this would be accomplished by separating the visible interface from the simulation code via some form of buffer. The accuracy of data calculated would vary depending on the number of simulation steps stored in the buffer. The importance of maintaining a given accuracy is illustrated by a video file on the attached CD that shows the effects of reducing the specified minimum accuracy on the particle net model mentioned above.

## References

- [1] D. Baraff, A. Witkin: Large Steps in Cloth Simulation (SIGGRAPH 1998)  
<http://www.pixar.com/companyinfo/research/deb/sig98.pdf>
- [2] G. Bradshaw, C. O'Sullivan: Sphere Tree Construction Using Dynamic Medial Axis Approximation (ACM, 2002)  
<http://isg.cs.tcd.ie/cosulliv/Pubs/SCA02.pdf>
- [3] J. Cameron: Physical Modelling using Geometric Algebra: Technical Milestone Report (2003)
- [4] C. Doran, A. Lasenby: Geometric Algebra for Physicists (CUP, 2003)
- [5] L. Herda, P. Fua, R. Plänkers, R. Boulic and D. Thalmann: Skeleton-Based Motion Capture for Robust Reconstruction of Human Motion (Proc. Computer Animation 2000, IEEE CS Press)  
<http://colab.epfl.ch/fua/papers/herda-et-al-ca00.pdf>
- [6] P. Jiménez, F. Thomas and C. Torras: 3D Collision Detection: A Survey (Computers and Graphics - vol. 25, 2001 - Elsevier)  
<http://citeseer.nj.nec.com/431815.html>
- [7] A.N. Lasenby, J. Lasenby, R. J. Wareham: A Covariant Approach to Geometry and its Applications in Computer Graphics (Technical Report, Cambridge University Engineering Dept. 2002)
- [8] A.N. Lasenby, C. Doran: Physical Applications of Geometric Algebra (Cambridge University, Lecture Notes, Part III, Physics, 1999)  
<http://www.mrao.cam.ac.uk/clifford/ptIIIcourse/course99/>
- [9] J. Lasenby, R. J. Wareham: By Personal Correspondance

---

<sup>39</sup>Note, it is also necessary to allow the time step to be fixed for recording of real time simulations.

- 
- [10] W. Press, S. Teukolsky, T. Vetterling, B. Flannery: Numerical Recipes in C++: Second Edition (Cambridge University Press, 2002)
  - [11] D. Sunday: Intersections of Rays and Segments with Triangles in 3D (May 2001 Algorithm, GeometricAlgorithms.com, 2001)  
[http://geometryalgorithms.com/Archive/algorithm\\_0105/algorithm\\_0105.htm](http://geometryalgorithms.com/Archive/algorithm_0105/algorithm_0105.htm)
  - [12] R. J. Wareham: Computer Graphics using Conformal Geometric Algebra (MEng Project Report, 2002)
  - [13] E. Welzl: Smallest enclosing disks (balls and ellipsoids) (Lecture Notes in Computer Science Vol 555, 1991)
  - [14] A. Witkin, D. Baraff, M. Kass: Physical Based Modelling (SIGGRAPH Course Notes 25, 2002)

All references available in digital form can be found on the attached CD.