GEOMETRIC MODELLING FOR REAL TIME FLIGHT SIMULATOR APPLICATIONS

David Sanders, Mark Jaques and Heather Clothier

ABSTRACT: Several geometric modelling methods are compared for use within real time flight simulation systems. Algorithms are presented which give a rapid solution for simplified versions of the models. The algorithms are robust and are suitable for Flight Simulator applications. The aircraft geometric structure is modelled as connected cylinders and the range of motion and the position of the aircraft is quantised. The programs take account of geometric and obstacle constraints. Although the solutions are sub optimal due to the simplified models used, the calculation time is short. The static model of the aircraft environment is held in computer memory as solid polyhedra. This model may be updated as new objects enter or leave the area of the aircraft. Overlapping spheres are used to model flying objects or moving objects on the ground. In this form the model data can be accessed efficiently by flight planning algorithms. The approach is global and uses a 3-D or 6-D graph of quantised configuration space. The method can also be applied to the planning of automatic low level flight path.

Introduction

In the 1970s "zero flight time" conversion was achieved and has worked successfully throughout the 1980s. Future flight simulation techniques must be effective in training pilots, but low cost, as the annual military budgets of European nations reduce and civil aircraft traffic and therefor profit drops. In the military sector, environmental and economic pressures may lead to a transfer of funding from field operations to simulated training. Partly for this reason, "Technology in the field of human factors, including simulation", has been identified as a common European priority area by the Independent European Programme Group (IEPG).

The MOD commissioned a training analysis study by Redifusion and Easams. The results are expected to suggest that three levels of simulator will be required [1]. A basic flight simulator for teaching pure aircraft handling, operating skills and instrument flying; a simulator to introduce the tactical use of the aircraft and its sensors and weapon systems; and a full simulator for mission practice and the development of tactics. The simulators are expected to be modular in architecture and supported by a range of cockpit and desktop trainers. The modelling methods discussed in this paper can be used in all but the most basic type of simulator.

The environment of an aircraft includes static and dynamic objects. The dynamic environment consists of the aircraft, and objects and aircraft to be engaged or avoided. The free space left available to the aircraft depends on the accuracy of the models used for this changing environment. As part of his dissertation in 1987, Balding[2] completed a study of modelling methods and considered the following as important requirements:

- Fast intersection calculations.
- Fast model generation.
- Efficiency (in terms of the volume occupied).
- Ease of use with path planning algorithms.
- Low memory storage requirements.

In this paper, past work described in [3] is examined and models are discussed with reference to the above requirements. The models considered can be divided into three categories, the Static Environment, the Aircraft and the Dynamic Obstacles. These categories are discussed in the following sections.

The Static Environment.

Of the modelling methods listed and discussed in [3], the following were investigated for the static environment, these were Polyhedral models, Constructive solid geometry models and Surface models. Most published computer models of aircraft surroundings take the form of polyhedral obstacles with flat surfaces and straight edges. These models are difficult to deal with in real time applications and calculation is slow. If both the aircraft and obstacles are modelled by polyhedral shapes then the accuracy is high but computation time is extended.

David A Sanders is a Senior Lecturer and Mark Jaques is a Researcher in the Automation, Robotics & Manufacturing Research Group in the School Of Systems Engineering, Portsmouth Polytechnic (UK) PO1 3DJ. Tel: (0705) 842565, FAX: (0705) 842351. Heather Clothier is a lecturer in Computer and Management Science at Sheffield City Polytechnic.

Constructive Solid Geometry represents conglomerations of objects as ordered binary trees. Terminal nodes are either primitive leaves which represent solid primitive shapes, or transformation leaves which contain the defining arguments of rigid motions. Non-terminal nodes represent operators such as rigid motions, intersection, difference or regularized union. More information about CSG representations may be found in [4] and [5].

Surface modelling methods have been used to model complex surfaces in detail. An introduction to surface modelling is given in [6]. Surface modellers use complex parametric functions such as Bezier equations to represent the detail of surfaces. These representations are difficult to use for intersection checking as only surfaces are represented. It is difficult to determine whether a point in space is inside an obstacle or not and consequently, to decide whether surfaces intersect is also difficult.

The Aircraft

The requirements for the aircraft model are similar to those for the dynamic obstacles described in the next section. At Link-Miles, control loop calculations have been reduced to approximately 130mS from input to visual update using CCC 3280 processors [7]. For future real time flight simulator applications, an important factor will be the speed of intersection calculation. A constraint was that the aircraft model selected needed to contain the entire volume of the aircraft. Aircraft have a similar design in that they have three major links, (the body and two wing sections) and three major axes (Roll, Pitch and Yaw). The simplest possible representation was three lines. Constant distances from the lines were then defined as enclosing the outer casing of the aircraft. This gave three connected cylinders as shown in figure 1. The advantages of this representation were that the cylinders modelled the aircraft body efficiently and the intersection calculations between the aircraft and obstacles were simple. The calculations simply consisted of:-

(i) In the case of a sphere, finding the distance from the centre of a sphere to the closest point on a line. From this distance was subtracted the radius of the arm and the sphere, to give the distance between the arm surface and the sphere surface.

(ii) In the case of other obstacle models, the obstacle model was expanded by the radius of the cylinder and the calculation reduced to comparing the position of the centre line with the obstacle. This was similar to the 'growing' techniques described in [8].

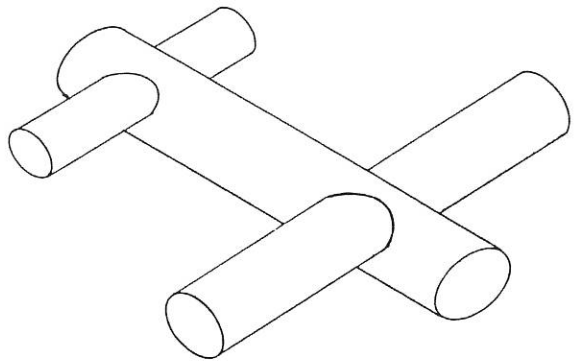

Figure 1: The aircraft model

Dynamic Obstacles.

Spheres are the simplest three dimensional shapes and Hopcroft et al [9] described how to calculate intersections among spheres efficiently. These calculations were easily modified to deal with the intersection between lines and cylinders required for the aircraft model. The method of modelling dynamic obstacles by spheres was initially selected for use in this research. Any shape may be modelled by spheres to any accuracy. The greater the accuracy required however, the larger is the number of spheres needed. Experimental work demonstrated that for larger numbers of spheres the computation time increased so that the accuracy of a model was limited by the computation time permitted for the path finding algorithm. This is described further in later sections.

In general, it is difficult to decide on the best sizes and positions of spheres to model real obstacles. In practice the number of spheres used to model obstacles were 1, 2 and 4. This made the models simple and speeded up intersection calculations, requiring little computer storage. This initial work with sphere models was used to compare intersection calculation speeds for several other models of the dynamic obstacles. Two other models compared favourably:- similar 2-D slices in configuration space - and - Six Sided Parallelepiped. The 3-D obstacle shapes considered during the work described in this paper were cylinders and simple six sided polyhedra. Although the algorithms for sphere calculation were potentially simple, the parallelepiped or similar 2-D planar slices tended to model these 3-D shapes as accurately. The method using 2-D slices is described in [3].

The Transformation into Configuration Space

Once obstacles had been modelled, the data was processed to transform the obstacles into a configuration space. A point obstacle in cartesian space is not transformed into a point in configuration space. It is transformed into one or more complex three dimensional shapes. Complex shapes may be represented within a computer as geometric shapes, units of space or by approximating the shapes by mathematical curves. The method selected for the work presented in this paper represented the obstacles as regions of configuration space consisting of small units. The method was not restricted to any particular design of aircraft and may be used with any number of degrees of freedom.

A graph was created which consisted of a six dimensional structure of unit regions. The 6-D graph had each dimension corresponding to a position, Θ_1, Θ_2 and Θ_3, or a principal degree of freedom of the aircraft, Θ_4, Θ_5 and Θ_6. Each unit was initially set to 'CLEAR' status and the positions (in configuration space) at which the aircraft intersected obstacles were then calculated. Each unit represented a range of positions for the aircraft, in terms of, ($\Theta_1$cent, $\Theta_2$cent ...$\Theta_6$cent), plus a degree of movement away from these central values, and a range of configurations for the aircraft, in terms of, ($\Theta_4$cent, $\Theta_5$cent, $\Theta_6$cent), plus a degree of movement away from these central configuration values. All units together represented the whole aircraft work-space and the number of units in the graph, $Node_{Total}$, was given by:

$$\frac{(\Theta_1 max - \Theta_1 min)}{2 \times \delta\Theta_1} \times \frac{(\Theta_2 max - \Theta_2 min)}{2 \times \delta\Theta_2} \times \frac{(\Theta_3 max - \Theta_3 min)}{2 \times \delta\Theta_3} \times \frac{(\Theta_1 max - \Theta_1 min)}{2 \times \delta\Theta_1} \times \frac{(\Theta_2 max - \Theta_2 min)}{2 \times \delta\Theta_2} \times \frac{(\Theta_3 max - \Theta_3 min)}{2 \times \delta\Theta_3}$$

where $\Theta_n max$, $\Theta_n min$ = the upper and lower limits of Θ_1.

This will later be expressed as:

$$\prod_{j=1}^{6} \frac{\theta_j max - \theta_j min}{2 \times \delta\theta_j}$$

If at any configuration in a unit, the aircraft intersected an obstacle, then the unit was set to BLOCKED. If at all configurations in a unit the aircraft did not intersect an obstacle then the unit remained CLEAR. The problem for an aircraft route planner is then reduced to checking a series of neighbouring units between the START and GOAL configurations that are CLEAR.

The first method considered for transforming obstacles into configuration space was to check each unit of the graph for intersections with each obstacle. This method was slow, taking up to fifteen minutes to calculate for a point obstacle. The program running time was proportional to both the number of obstacles and the number of nodes. If the free space was assumed to be larger than the blocked space, a faster method was to consider each obstacle and test for the nodes which could contain the transformed obstacle. This was the method adopted and the algorithm was as follows: *For a node in the graph where the aircraft could intersect the obstacle, recursively test all the neighbouring units to see if they are also within the reach of the aircraft.*

Transforming spheres into a configuration space.

The graph data structure was initialised. As the graph carried out intersection checks at a limited number of positions, only a limited number of trigonometric solutions were required and these were calculated at the start. Before the obstacles were calculated all the units in the graph had a flag set to 'CLEAR' status. Four other flags were used with each node and these were:- *'New obstacle'* *'Rear wing tested'* *'Forward wing tested'* *'On list'*

Each unit code was stored as one byte of computer memory in the array *NodeStatus* and the flags used one bit each. The environment and obstacle data were retrieved from files and the first task for the program was to read this data. The blocked space was then calculated. A configuration was calculated at which the aircraft was close to the sphere centre. If the main body was being considered, then the configuration where the tip was at the centre of the sphere was calculated. This configuration was the first unit for the transformed obstacle. This gave a starting configuration.

The first configuration was set to BLOCKED . Its neighbouring units were also tested and if they were set to BLOCKED then their neighbours were checked. The position problem was solved using the forward kinematic calculations and the minimum distance between the obstacle and the aircraft was calculated. The method continued recursively until the whole obstacle transformation was completed. All units were set to BLOCKED, which had any two opposite neighbouring units which were also BLOCKED. Any units which were on the edge of the now solid obstacle were recorded on a list. All the neighbours of the units on the list were tested, and the process

repeated until the surface of the transformed sphere was completely defined. Nodes which were BLOCKED were stored on a list of units to be expanded later. When a unit was expanded it was retrieved from the list and new BLOCKED points were added to the list. When all the nodes on the list were exhausted the obstacle transformation was complete. The operation of the lists, the testing routines, the expand routines and fill in routines are described in [3]. The most important consideration was processing speed. Times for calculating obstacles were recorded during the project and these are presented in the results section of this paper.

Other Models.

Although the fastest global transformations were achieved using the solid sphere and 2-D slices, several other models were investigated in order to compare performance. These were:- (a) Semi Solid Spheres, (b) Hollow Spheres, and (c) Simple Polyhedral Shapes.

(a) *Semi Solid Spheres*. The method was similar to that used for the solid sphere except that the expansion routine expanded two nodes at a time from the centre of the sphere. When a CLEAR node was reached, the node was placed onto a spare list (list 3) along with the node it had expanded from. Once all the expansions had revealed CLEAR nodes, a subroutine *ExpandIn* tested the node pairs on list 3 to see if a collision occurred between them. When a collision occurred this was taken to be the edge of the sphere and the *NodeStatus* was set to collision, otherwise the node was assumed to be on the edge of the sphere.

(b) *Hollow Spheres*. This program effectively followed the surface of the sphere, setting the surface nodes to BLOCKED so that the transformation programs would be unable to enter the sphere. Instead of beginning the process at the sphere centre, the Z coordinate was set to the top of the sphere. The program then expanded the nodes as described above, placing BLOCKED nodes onto the list. If more than six collisions occurred in a single expansion then the test node was inside the sphere and the collisions were removed from the list. Where less than six collisions had occurred the program was following the edge. The CLEAR nodes recorded were assumed to represent the edge of the sphere when passing the data to the path planning computer. This method was slow as every time the program entered the sphere, six nodes were tested and removed from the lists. The sphere models were more complex in configuration space and following the surface was a complex task.

(c) *Simple Polyhedral Shapes*. The method modelled the obstacles as six sided parallelepiped and the program established the position of the edges of the model in X,Y & Z.

Results.

The most important consideration for the system was that it should be suitable for real time applications. Times for transforming obstacles were recorded during the project and as an example, the times for the different models to transform the vertical cylinder into configuration space are shown below. The times were recorded with the Z axis of the cylinder at X =0 mm, Y =3000 mm and Z =5000mm with respect to the origin.

Model	Time (Seconds)	Number of BLOCKED nodes recorded.
One Sphere	3.2	2397
Two Spheres	5.2	2301
Hollow Sphere	8.6	2409
Simple Polyhedron	9.7	1984
2-D Slices	2.0	2494

Figure 2: Table of Transformation times for an Upright Cylinder in space.

The Sphere Model: An obstacle was modelled first as a single sphere of the smallest radius which would enclose the obstacle. If time allowed it was modelled by two smaller spheres and then four spheres. Nodes set to BLOCKED associated with the first sphere tested usually also collided with other spheres. The forward kinematic solutions did not need to be recalculated for these nodes but the total calculation time increased with the number of spheres because the overhead of calculation for each sphere was greater than the saving in time achieved as the spheres became smaller. This meant the single sphere calculation was faster than the calculations for multiple spheres although the single sphere model was less accurate and had a larger volume.

The problem when using more than one sphere was that the centre of several spheres would be set to BLOCKED (with some surrounding nodes) after the expansion of the first sphere. As these nodes were BLOCKED, later

spheres were sometimes not retested so that many nodes were not added to the list. To overcome this problem the centre was tested for an old collision. If TRUE then the node was placed on a new list, (list 3) where the node was expanded later. The nodes on the new list were then dealt with until the list was empty, meaning that for models using two spheres all the nodes outside the first sphere had been found. A subroutine *Expandtest* tested each node after expansion to see if a collision had occurred with the first sphere. If it had then the node was added to list 3 to be expanded at a later date and a bit was set in the flag *NodeStatus* so that the node was not retested. If the node was CLEAR then the edge of the first sphere had been found and the node was tested for collision against the new sphere using a subroutine *Testpos*.

Discussion and Conclusions

For the transformation methods a graph of calculation time vs. discrete work-space volume can be expected to be linear. The calculation time for an obstacle was approximately proportional to the number of units tested, the total number of nodes being the work-space volume. The computer time required for obstacle transformations was short. The initial conversion time to model the static environment was slow; Up to twenty five minutes of computer time depending on the complexity of the model, but the transformation was only performed when the system was powered up. High accuracy models required more computation time and therefore longer solution times. Low accuracy models required links or obstacles to be oversized to eliminate the chance of undetected collisions. Lowering the accuracy led to the rejection of valid solutions. The world may be modelled accurately by Polyhedral, CSG or surface modelling methods but they are complex models requiring complex intersection calculations. The generation of these models was slow. The transformation of the static environment need only be made once, so that computation time is not a problem. An accurate model was therefore selected and Polyhedra were used to model the static environment. For later work using a simple static environment the model was reduced to a single polyhedral shape modelling the work surface. For dynamic models, speed of calculation was important. The simplest possible intersection calculations for the local methods were made using the sphere model. Calculation was reduced to finding the distance from the aircraft to a point and subtracting the radius of the sphere to give the distance to the surface of the sphere.

Modelling with more than one sphere was considered. As the real environment becomes more complex so more spheres are needed for the model. It was considered how increasing the number of spheres might increase the accuracy of the model. The case of modelling a unit cube was investigated by Balding [2]. A cubic number of spheres was used and the spheres formed a regular pattern and were equal in size. An infinite number of spheres was required to model a cube completely but modelling objects using the same sized spheres was inefficient. For example, in modelling a cube using sixty-four spheres of the same size, eight of the spheres are totally enclosed and might easily be replaced by a single larger sphere without increasing the model volume.

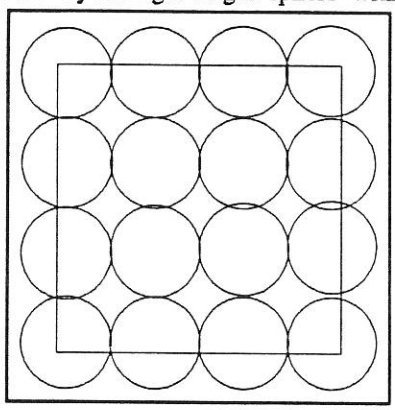

Figure 3.a: Cross Section of a cube modelled by 64 spheres

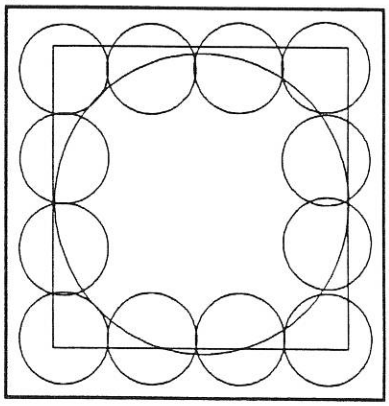

Figure **3.b:** Cross Section of a Cube modelled using different sized spheres.

To compare the modelling of obstacles using single and multiple spheres, as an example, a model of a cylinder using one and two spheres is compared. The volume of two spheres of radii 35 mm were compared to that of one sphere of 70 mm as shown below. The volume of Two Spheres is $2 \times 4/3 \times \pi \times 35^3 = 359{,}188\ mm^3$ and the volume of One Sphere is $4/3 \times \pi \times 70^3 = 1{,}436{,}755\ mm^3$. The area of the two spheres would be much smaller except that the model of the aircraft must then be considered to find the union volume, Aircraft U Model. The Union Volume of a single sphere is $4/3 \times \pi \times 150^3 = \underline{11{,}494{,}040} mm^3$, and the Union Volume of Two Spheres is $2 \times 4/3 \times \pi \times 115^3 = \underline{12{,}741{,}211}\ mm^3$

There were a similar number of collisions for both models. When points within the second sphere were not tested to see if they had collided during the calculations for a previous sphere, this partially explained the lack of

improvement in processing time for the model using two spheres. When the points within a sphere were tested to see if they collided with a previous sphere a saving in processing time could have been expected but in the routine *ExpandTest* the nodes were continuously expanded until they reached the outer surface of a sphere where tests where conducted to see if the outer surface node collided with another sphere, before the next sphere had been filled. This wasted processing time.

Considering the simple six sided parallelepiped model, the volume of the model for the horizontal cylinder was less than that of both a single sphere or multiple spheres. Parallelepiped Volume $= (60 + 160)^2 \times (140 + 80) = 10{,}648{,}000\,mm^3$, (where 160 was included due to the aircraft. This was added to allow for the expansion required in X,Y and Z). This potentially reduced the number of blocked nodes (and therefore the processing time), but the shape and therefore the calculations were more complex so that calculation time increased. Using the two dimensional slice model of the cylinder, Θ_2 and Θ_3 were only determined for a single slice. This reduced the processing time as this slice of BLOCKED nodes was copied for all Θ_1 within the bounding base configuration angles. The calculation time for complex obstacles modelled as spheres was short as each sphere required only four data items, (three cartesian coordinates and the sphere radius). For use with the Global Planning system, the static environment was modelled accurately as several polyhedra and was transformed into configuration space before introducing dynamic obstacles. As this transformation took place once, at the beginning of the program, there was no time constraint.

The aircraft geometric model consisted of three lines connected across the body surrounded by a skin a constant distance from this skeleton. This model was simple and proved to be fast. Several different models were considered for the dynamic obstacles and two were selected as they performed the transformation into configuration space in the fastest times. A local route planner performed faster when using spheres compared with the other models. It should be noted that there must be a point at which increasing the number of spheres, in order to increase the accuracy of a model, becomes impractical and at this point the Data Processor could change to use a polyhedral model. An extension of these models would be to model the aircraft as a 2-D slice in configuration space, that is effectively a short cylinder as shown in figure 4. This could reduce the configuration space to 5-D or even to 4-D if a sphere was considered.

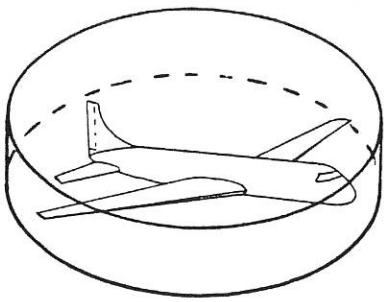

Figure 4: Aircraft modelled as a short cylinder

References

[1] **Fitzsimons B**, "*Post gulf simulation*", Aerospace - The Royal Aeronautical Society, Nov 91, pp 8-10. **1991.**

[2] **Balding N.W.** 1987(PhD) "*Real Time Path Planning for Robot Arms*", Durham (UK). **1987**

[3] **Sanders DA.** 1990(PhD) "*Automatic Robot Path Planning With Constraints*", Portsmouth (UK). **1990.**

[4] **Braid I.C** "*The Synthesis of Solids bounded by many Faces*", Communications of the ACM, Vol 18, pp 209 - 221, **1975.**

[5] **Requicha A.A.G**"*Constructive Solid Geometry*", Rochester University, NY Production Automation Project, Nov 77. NSF/RA-770476, **1977.**

[6] **Ball** "*Designers guide to shapes*", CADCAM Int, Mar 83, pp 32-34 and Oct 83 pp 42 - 44, **1983**

[7] **Cooper G**, "*Tomorrows cockpit displays*", Aerospace - The Royal Aeronautical Society, Nov 91, pp 12-14. **1991.**

[8] **Udupa S.M** 1977(PhD) "*Collision detection and avoidance in computer controlled manipulators*". **1977**

[9] **Hopcroft et al** "*Efficient detection of intersections among spheres*", Int J of Robotics Research, Vol 2-4, pp 77 - 80, **1983.**