Fast and Accurate Spacio-temporal Intersection Detection with the GJK Algorithm

Kevin Vlack, Susumu Tachi
University of Tokyo, Graduate School of Information Science and Technology
Department of Information Physics and Computing
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656, Japan
vlack@star.t.u-tokyo.ac.jp, tachi@star.t.u-tokyo.ac.jp

Abstract

This paper discusses an *extrusion* technique to quickly and robustly tackle the four-dimensional problem of spacio-temporal intersection detection for rigid bodies with arbitrary motion. We combine the GJK algorithm with Brent's method to determine non-collision or time-of-impact (TOI) for an object pair over a complete time interval. Experimental results show this method capable of supporting accurate rigid body simulation of large scale and highly dynamic environments at interactive frame rates on a common PC.

Key words: collision detection, GJK algorithm, extrusion, rigid body simulation, real-time animation

1. Introduction

It is common knowledge that solid objects do not interpenetrate in reality. The representation of this basic principle is paramount to the feasibility of a model of the physical world. Further, the non-penetrability constraint is quite intuitive, so it can appeal to our spatial cognition processes to improve our sense of presence in virtual reality applications, as well. Enforcing this constraint requires the identification of when and where objects begin to interpenetrate, and in the setting of physical simulation, this is known as the problem of collision detection.

Collision detection research has advanced significantly in recent years, motivated primarily by the interest in physically based modeling of virtual environments. It's an intricate endeavor, because collisions are discontinuous events, and special care must be taken to find them when modeling an otherwise continuous world. We can calculate these events analytically for relatively simple environments such as a racing track or a billiards table, but these methods are traditionally confined to highly constrained applications such as video games, and an analytical solution is intractable for the general case. A more robust physical simulation will typically detect collisions numerically through the evaluation of pair-wise intersection tests over time. Cameron [1] classifies these intersection tests into three distinct categories: static, sweeping, and extrusion.

Static intersection tests check for intersection at specific instances in time. These tests are relatively simple and fast, and never report false alarms. However, they often miss collisions, especially for small or flat objects †, because although an object pair might not intersect at two separate times, they may have actually passed straight through each other (Fig 1 center). These misses are blatant errors that lead to infeasible simulation behavior. When objects are interdependent, the problem worsens as these errors accumulate – the simulation becomes increasingly unrealistic, commonly resulting in eventual breakdown due to unrecoverable failures.

Sweeping intersection tests detect the intersection between the sweeping volumes of objects over time, or an approximation thereof. Although these tests never miss, they also conservatively report false alarms that never actually occur, because they do not account for the displacement of objects as they move (Fig. 1 right). This overshooting can be resolved by recomputing the sweeping volumes for intermediate time steps to verify whether there was an actual collision, but the computation required to do so may be prohibitive. Consider a stack of blocks in freefall – even over miniscule time intervals their swept volumes intersect, although the blocks themselves never come into contact.

Extrusion intersection tests take a four-dimensional

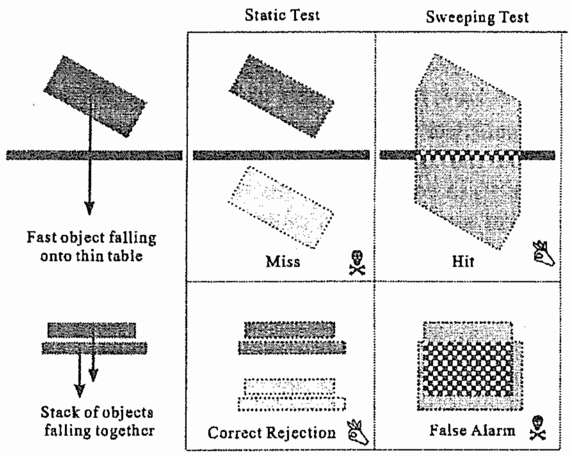

Fig. 1 Examples of intersection tests over time

† For brevity, "object" refers to a geometric primitive such as a box, sphere, triangle, convex polyhedra, etc.

approach to detect spacio-temporal intersection directly, without misses or false alarms. The mathematical basis of extrusion is straightforward, consisting of distributive set operations on points in space-time, but the actual boundaries of these 4-D sets can be quite complex, especially when objects rotate. Fortunately, the intersection of two object extrusions can be detected without ever constructing their boundaries explicitly, as we will discuss below in section 3.

A fourth consideration consists of detecting intersection in *configuration space*[2], but this generally concerns the much harder problem of collision avoidance, and will not be addressed further in this paper.

2. Overview of the GJK Algorithm

We begin with an overview of the GJK algorithm, which is a classic method to find the closest distance between two convex polyhedra. This is a very brief sketch, and the reader is referred to [3, 4, 5, 6] for a more in-depth discussion, and several techniques to optimize its performance. We introduce how GJK can be used for the spacio-temporal case in the next section.

2.1 Supporting Vertex

For a convex polyhedron **X** and a vector $\vec{s} \in \Re^3$, assume there exists a routine $S_X(\vec{s})$ that returns the *supporting vertex* of **X** in the direction $\vec{s}$, defined as:

$$S_X(\vec{s}) \in \text{vert}(\mathbf{X}) \text{ where } \vec{s} \cdot S_X(\vec{s}) = \max\{\vec{s} \cdot x : x \in \text{vert}(\mathbf{X})\}$$

Typically, this vertex is unique, but not necessarily. It can be found efficiently with a technique called *hill climbing*, which uses adjacency information of each vertex to search for a local (and also global) maximum [5]. Further improvements such as using hash tables to achieve nearly constant search time are explained in [6].

2.2 Calculating closest distance

Begin GJK with two convex polyhedra **P** and **Q**, an arbitrary nonzero vector $\vec{s}_0$, and two vertex sets V_P and V_Q, initially empty. At each iteration *i*, append the vertices $S_P(\vec{s}_i)$ and $S_Q(-\vec{s}_i)$ to V_P and V_Q, respectively. Then calculate the feature of the Minkowski sum $V_Z = V_Q - V_P$ (note the minus sign) which is closest to the origin. Refine V_Z to contain only this feature, and since it must be either a vertex, edge, or triangle (in 3D), V_P and V_Q will always contain three vertices or less. The length of the vector z from the origin to the closest point shall be referred to as the *Minkowski distance* (MD), and GJK reiterates with $\vec{s}_{i+1}$ set to z until this distance no longer decreases. If **P** and **Q** are disjoint, convex analysis proves that the Minkowski distance will converge monotonically to the global closest distance, and the closest features can be interpreted directly from the final sets V_P and V_Q. Figure 2 illustrates an example execution of GJK, with sets V_P and V_Q on the left, and the corresponding Minkowski sum $V_Q - V_P$ on the right.

Empirically, GJK completes within a constant number of iterations (generally less than eight), nearly independent of the geometrical complexity of **P** and **Q**. Further, if the states of **P** and **Q** do not change much between queries, the work from the previous query (e.g. V_P and V_Q), can be used to reinitialize the current query in order to exploit temporal coherence. This enhanced version of GJK executes extremely quickly, typically completing within one or two iterations on the average, and ranks among the most competitive closest feature algorithms reported to date. GJK is especially attractive for its simplicity, since it only requires simplex object data, which is very scalable and easy to build. [7]

2.3 Supporting distance and static intersection

GJK also serves as an efficient static intersection test when it calculates the *supporting distance* between **P** and **Q** in the direction $\vec{s}_i$ at each iteration (see fig. 3). This is conceptually different from the Minkowski distance mentioned above, and is defined as:

$$SD(\mathbf{P}, \mathbf{Q}, \vec{s}) = \vec{s} \cdot S_Q(-\vec{s}) - \vec{s} \cdot S_P(\vec{s})$$

which is the *signed* distance between the supporting planes of **P** and **Q** in opposing directions of $\vec{s}$. Unlike the Minkowski distance, the supporting distance sporadically increases and decreases at each iteration of

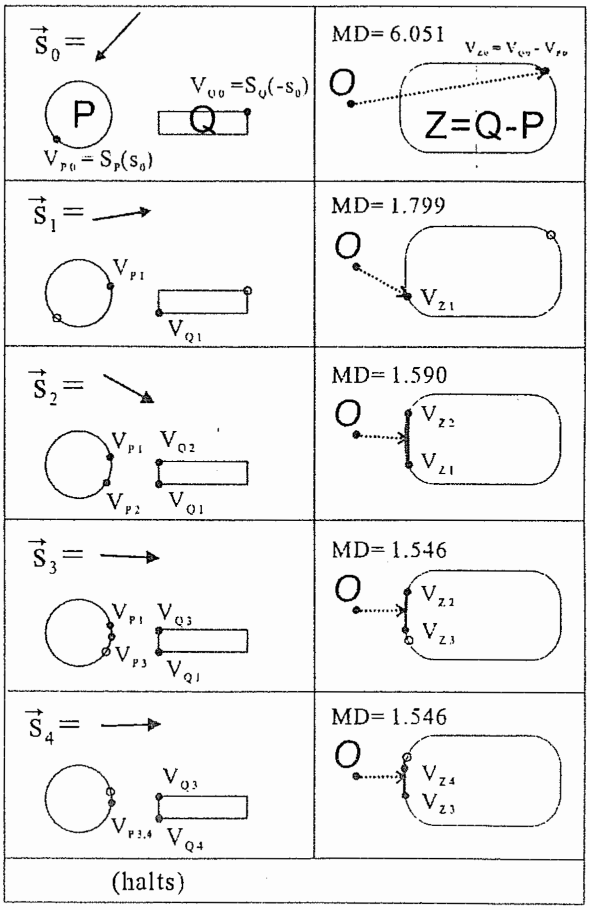

Fig. 2 Example 2D execution of GJK

GJK. If this distance is positive, then $\vec{s}_i$ is called a *separating vector* because it defines a plane in space that separates **P** and **Q**. In the example above, $\vec{s}_1$ through $\vec{s}_4$ are all separating vectors. A boolean query can stop once it finds the first such vector, because it proves **P** and **Q** are disjoint. If the GJK algorithm finds a local minimum for the Minkowski distance and halts before a separating vector can be found, then such a vector provably doesn't exist, and **P** and **Q** therefore must either intersect, or be so close together that round-off error dominates the computation. GJK is numerically sensitive for this reason, but the inherent problems can be alleviated in most cases by enforcing separation between objects by a nonzero *collision tolerance* based on floating point precision.

3. Separating Hyperspace Test

Consider the space defined by all separating vectors between $\mathbf{P}(t_0)$ and $\mathbf{Q}(t_0)$. Geometrically speaking, if **P** and **Q** intersect at t_0, this space is null, otherwise it is an open pyramid extending away from the origin. The spacio-temporal method we introduce here is an extrusion test over the time interval t_0 to t_1, but instead of calculating the extrusions of $\mathbf{P}(t)$ and $\mathbf{Q}(t)$, we analyze the extrusion of their separating space over time, referred to here as their *separating hyperspace*. To detect intersection over t_0 and t_1, we determine whether the separating space ever vanishes in the interim by testing whether the spaces at t_0 and t_1 are disjoint.

This is based on the fact that in the case of linear motion, i.e. no rotation or acceleration, **P** and **Q** never intersect if and only if the their separating hyperspace is a convex set. In the case of nonlinear motion it is only semi-convex, and if **P** and **Q** intersect, the separating space at

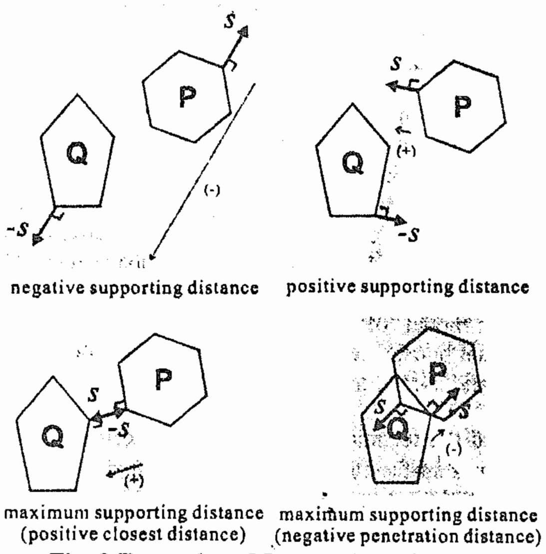

Fig. 3 Examples of Supporting Distance

t_0 may vanish undetected, and a subset of which could reappear at t_1. However, these errors are the result of *second order* motion, which is practically negligible for sufficiently small time steps, such as for animations of objects that do not rotate or accelerate too quickly.

With this in mind, our proposed method is the following. Assuming by induction that $\mathbf{P}(t_0)$ and $\mathbf{Q}(t_0)$ do not intersect, we execute the GJK algorithm at t_0. At each iteration, we compute $\mathrm{SD}(\mathbf{P}(t_0), \mathbf{Q}(t_0), \vec{s}_i)$. If this distance is positive, $\vec{s}_i$ is a separating vector at t_0, so we compute $\mathrm{SD}(\mathbf{P}(t_1), \mathbf{Q}(t_1), \vec{s}_i)$. If this distance is also positive, then $\vec{s}_i$ is a separating vector for both t_0 and t_1 and their separating spaces intersect, so the test returns non-collision for the entire time interval.

This extrusion test fails if the GJK algorithm halts before it finds a separating vector for both t_0 and t_1, but that does not necessarily imply collision for two reasons: GJK might not have encountered such a vector in the relatively few iterations before halting, and the separating space for **P** and **Q** may have evolved over time without ever vanishing. So, like the sweeping test, we must further investigate with intermediate time intervals, as described in section 5. However, this extrusion test combines the complementary merits of both static and sweeping tests, and performs better for a broad scope of rigid body motion (Fig. 4).

4. Maximum Supporting Distance and TOI

Consider the maximum $\mathrm{SD}(\mathbf{P}, \mathbf{Q}, \vec{s})$ at a particular instant in time for all vectors $\vec{s}$ on the unit sphere. A positive value is equivalent to the closest distance when **P** and **Q** are disjoint, and a negative value is the penetration distance when they intersect (Fig 3). If we represent the maximum supporting distance between $\mathbf{P}(t)$ and $\mathbf{Q}(t)$ as a continuous scalar function of time, we can identify collisions by checking whether or not this function drops below zero, and then determine their

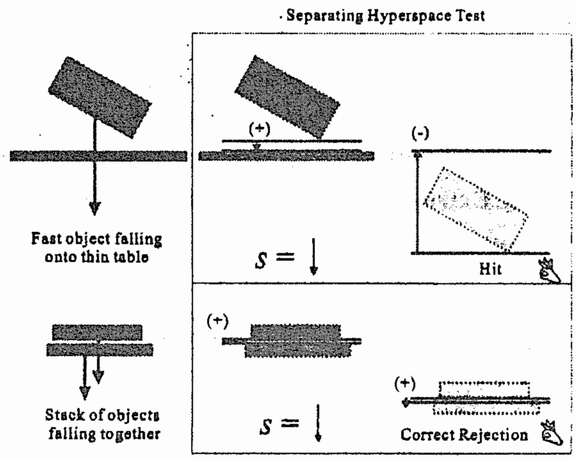

Fig. 4 Extrusion tests robustly give correct results

exact TOI with a root solver.

Unfortunately, this is a difficult task in practice.

This function's behavior is fairly complex in general, especially because its first derivative is discontinuous for objects such as polyhedra, whose closest features change instantaneously. In addition, although a closest feature algorithm such as GJK finds the maximum supporting distance efficiently if $\mathbf{P}(t)$ and $\mathbf{Q}(t)$ are disjoint, global optimization based on convex analysis breaks down if $\mathbf{P}(t)$ and $\mathbf{Q}(t)$ intersect, and an exhaustive search may be required to compute its exact value.

One popular way to avoid these issues combines a closest distance algorithm with an analytical TOI estimator, which calculates the earliest possible time of impact for object pairs, based on their current closest distance and general equations of motion [8]. Objects are advanced forward at time steps that guarantee non-intersection, until the closest distance between an object pair falls below the collision tolerance. This ensures accurate collision detection, and works well for simple environments, but there are three major drawbacks.

First, these equations are complicated and must consider a multitude of variables, such as linear and angular velocity, acceleration, and object dimensions. Overly conservative predictions will converge poorly, and a robust implementation has a steep learning curve that could confound all but the most diligent developer.

Second, tracking the closest distance between two objects is a wasteful procedure if we only care about if and when they collide. As described earlier, a boolean intersection test may execute several times faster than a complete closest distance query, and with less overhead. Finding the closest distance for nonconvex objects is even more difficult, and quickly degrades performance.

Third, when we consider more than two objects whose collisions are interdependent, this one-sided approach requires a "TOI heap," which maintains the earliest time in which object pairs *might* collide. For environments consisting of hundreds of objects, this could result in a very large number of pairs, even after they are pruned by bounding volumes. Heap operations aren't free, and the effort required to update the heap could create a computational bottleneck.

5. GJK Extrusion Test with Brent's Method

The method proposed here takes a two-sided approach to compute the TOI retroactively, by approximating the (negative) penetration distance at t_1 as the maximum value of $\mathrm{SD}(\mathbf{P}(t_1), \mathbf{Q}(t_1), \vec{s}_i)$ among those we calculated before the GJK query halted. Since we only calculate $\mathrm{SD}(\mathbf{P}(t_1), \mathbf{Q}(t_1), \vec{s}_i)$ when $\vec{s}_i$ is a separating vector at t_0, this heuristic effectively estimates the time that the subset of separating vectors found at t_0 will vanish.

Our implementation uses Brent's method, which is a root solver that cleverly combines bisection, the secant method, and inverse quadratic interpolation to guarantee superlinear convergence [9]. If the initial extrusion test at t_0 and t_1 fails, we use the closest distance at t_0 and the approximate maximum supporting distance at t_1 to calculate an intermediate time t_x using Brent's method. We then repeat the extrusion test for the interval t_0 to t_x, but since the work for executing GJK at t_0 is identical, we can speed up this intermediate query by storing all the separating vectors $\vec{s}_i$ we found at t_0, and use this set (a constant number, generally less than seven) to recalculate $\mathrm{SD}(\mathbf{P}(t_x), \mathbf{Q}(t_x), \vec{s}_i)$ and obtain a new approximate maximum supporting distance at t_x. If this distance is negative, we continue normally with further iterations of Brent's method. However, if this distance is positive, we assume $\mathbf{P}$ and $\mathbf{Q}$ are disjoint from t_0 to t_x, so we recompute a complete GJK extrusion test for t_x and t_1 in order to determine the exact closest distance at t_x before we proceed.

The solver returns either the TOI t_x if its maximum supporting distance is positive and smaller than the collision tolerance, or noncollision if it finds a "photo finish", in which the extrusion test fails for the complete time interval t_0 to t_1 but succeeds when performed on the sequence of intervals t_0 to t_{x1}, t_{x1} to t_{x2}, ... t_{xn}, to t_1. As mentioned in section 3, this could happen fairly often for two reasons: a separating vector for both t_0 and t_1 may exist but the GJK algorithm halted prematurely before finding it, or the separating space for $\mathbf{P}(t_0)$ and $\mathbf{Q}(t_0)$ evolved over time without ever vanishing.

The ability to determine non-collision or time of impact for a complete time interval is most convenient when we are considering multiple objects whose collisions are interdependent, because we can apply this extrusion intersection solver to all object pairs with a single pass, and then use Mirtich's very clever Timewarp algorithm [10] to efficiently resolve collisions in the proper order.

6. Experimental Results

This algorithm has been implemented and initial experiments have been conducted on a 500 MHz Pentium III with an NVIDIA TNT to analyze its effectiveness for large-scale rigid body dynamics simulation on a common PC. The static environment consisted of an aquatic plant fixed in space with 8640 triangles, and the dynamic environment consisted of a skeletal model of a human torso separated into its individual bones and vertebrae, with 80 nonconvex rigid bodies and 24,790 triangles in total.

We used the techniques in [11, 12] to decompose each nonconvex surface into convex polyhedra and generate a hierarchical representation of bounding volumes known as k-dops[13], with $k = 14$. Bounding volume tree queries used k-dop extrusion intersection tests to prune primitive pairs, and generalized frontal tracking [14] to exploit temporal coherence. We used Mirtich's approach [8] with RK5-4-7FM, a fifth order accurate Runge-Kutta method with adaptive step-size control, to numerically integrate an impulsive collision response, with dynamic friction coefficient $\mu_d = 0.4$ and collision restitution coefficient $e = 0.7$. Static contact was modeled through microcollisions, and a more robust hybrid implementation remains as future work.

The simulation began by dropping the torso from a random position. The collection of bones fell freely with constant acceleration until it shattered onto the plant. (Fig. 5) The simulation completed once all of the bones had fallen through the plant or settled to a steady state.

The simulation advanced 1/24 seconds at each frame, and the actual execution time for each step is presented in figure 6, including the time for Euler integration, bounding volume computation, collision detection at the polygonal level with the GJK extrusion test, impulse integration, collision ordering with Timewarp, and image display with OpenGL. The peak at $t \approx 2$ sec indicates the point when the number of interdependent collisions within the skeletal model was maximized due to the crash. The bones then scattered, and eventually settled into steady contact with the plant with an increasing number of microcollisions, indicated by the steady rise in execution time until the simulation's completion. The ratio of execution time to simulation time ranged between 1.0 and 7.9 for the entire simulation. For a collision tolerance of $\varepsilon^{1/2} \approx 0.0003$, where ε is the 4-byte floating point epsilon, we found the TOI solver completed in 2.2 iterations on the average, with a worst case of 43 iterations, and an average "photo-finish" false alarm rate of 70%.

Admittedly, this omnibus benchmark is not a very useful measure of collision detection efficiency. Regardless, from this preliminary investigation on a middle-grade PC by today's standards, we are optimistic that the method proposed here is an improvement over current methods of its kind by a significant margin, and a more rigorous comparative study is very near future work.

7. Conclusion

The physical feasibility of rigid body dynamics depends on accurate collision detection, which is challenging to compute in the precious milliseconds between successive animation frames if we are using physically based modeling to improve the sense of presence in virtual reality applications. However, after experimentation with the GJK extrusion method proposed here, our observation is that the prospects of accurate collision detection in real-time for large scale virtual environments are very good, and we eagerly encourage others to help continue the progress in this exciting field of research.

References

1. S. Cameron: "Collision Detection By Four Dimensional Intersection Testing," *IEEE transactions on Robotics and Animation,* 6(3): pp. 291-302 (1990).

2. T. Lozano-Perez. "Spatial planning – a configuration space approach", *IEEE Transactions on Computers*, C-32(2): pp.108-120 (1983).

3. E. Gilbert, D. Johnson, D. Keerthi. "A Fast procedure for computing the distance between objects in three-dimensional space", *IEEE Journal on Robotics and Automation*, vol 4: pp. 193-203 (1988).

4. G. Van Den Bergen. "A Fast and Robust GJK Implementation for Collision Detection of Convex Objects", *Journal of Graphics Tools* (1999).

5. S. Cameron: "Enhancing GJK: Computing Minimum and Penetration Distances between Convex Polyhedra", *IEEE International Conference on Robotics and Automation*, pp. 22-24 (1997).

6. K. Chung, "An Efficient Collision Detection Algorithm for Polytopes in Virtual Environments", M. Phil Thesis, University of Hong Kong (1996).

7. E. Levey, C. Peters, C. O'Sullivan. "New Metrics for Evaluation of Collision Detection", *Proceedings of the 8th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media* (2000).

8. B. Mirtich. "Impulse-based Dynamic Simulation of Rigid Body Systems", *PhD Thesis, University of Berkeley* (1996).

9. R. P. Brent. *Algorithms for Minimization without Derivatives* (Englewood Cliffs, NJ: Prentice-Hall), Chapters 3, 4 (1973).

10. B. Mirtich. "Timewarp Rigid Body Simulation", *Proceedings of SIGGRAPH 00*, pp. 193-200 (2000).

11. K. Vlack. "Real-Time Collision Detection for Nonconvex Polyhedra Using Convex Surface Decomposition", *Proceedings of the 6th Virtual Reality Society of Japan Annual Conference* (2001).

12. K. Vlack. "An Efficient Bottom-Up Method to Construct Bounding Volume Trees for Real-Time Collision Detection", *Proceedings of the 6th Virtual Reality Society of Japan Annual Conference* (2001).

13. J. Klosowski, M. Held, J.S.B. Mitchell. "Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPs", *IEEE Transactions on Visualization and Computer Graphics*, Vol. 4 (1998).

14. S. A. Ehmann, M. C. Lin. "Accurate and fast proximity queries between polyhedra using convex surface decomposition". *Tech. Report TR01-012, Department of Computer Science, University of North Carolina* (2001).

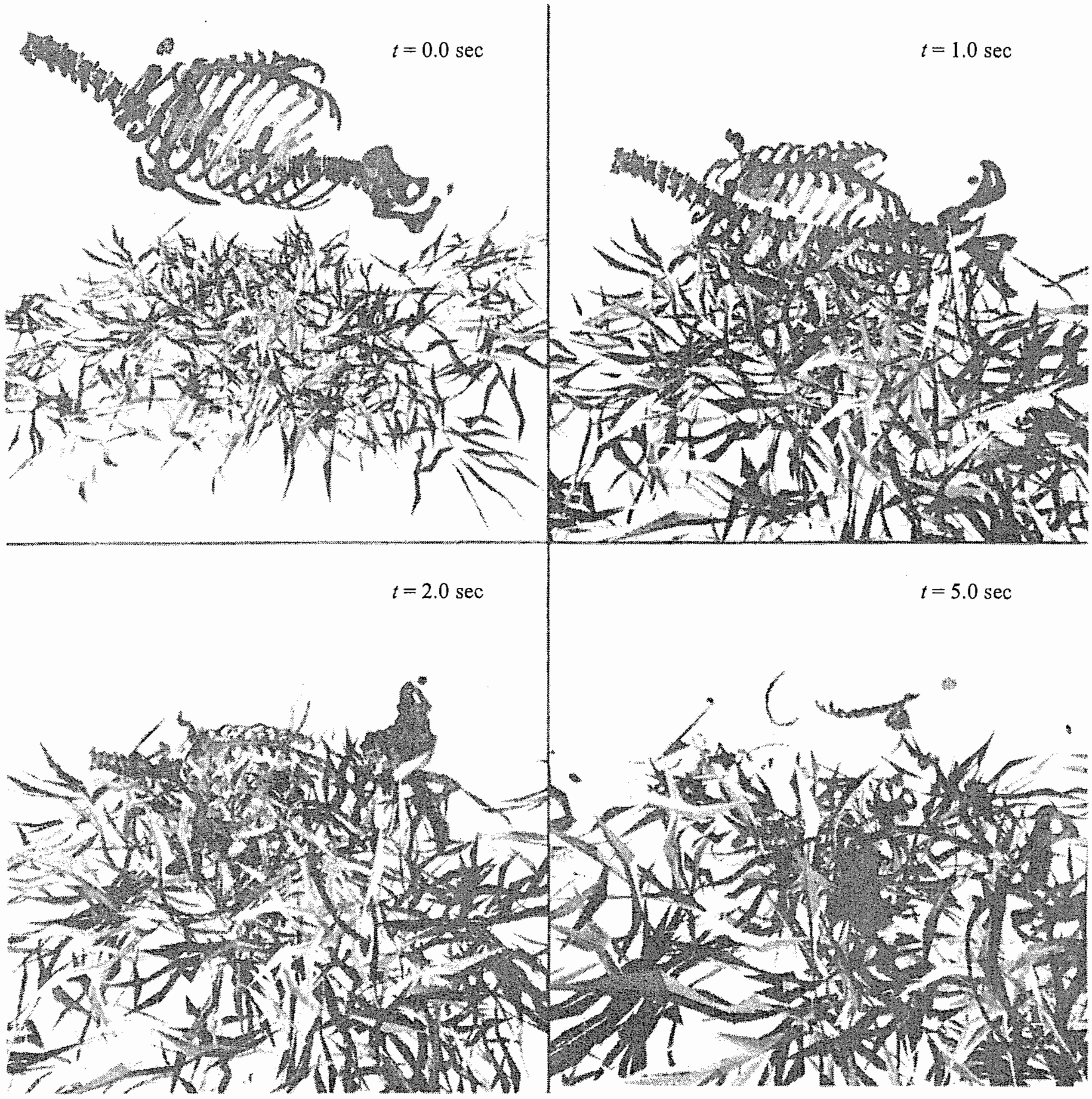

Fig. 5 Skeletal torso in freefall crashing into aquatic plant

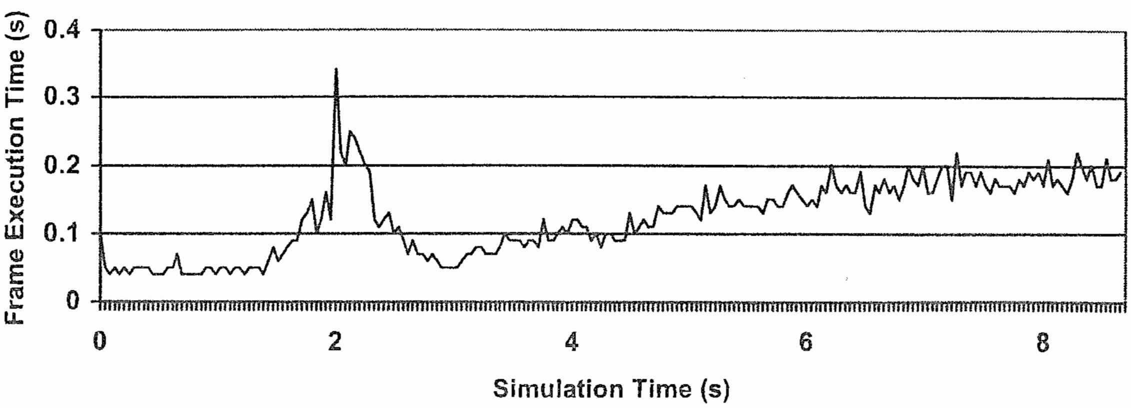

Fig. 6 Running Time for Torso-Plant Simulation