SAND98-2729J

Parallel Strategies for Crash and Impact Simulations

K. Brown, S. Attaway
Engineering Manufacturing and Mechanics Department
S. Plimpton
Parallel Computational Sciences Department
B. Hendrickson
Parallel Computing Sciences Department

Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-0443

Abstract

We describe a general strategy we have found effective for parallelizing solid mechanics simulations. Such simulations often have several computationally intensive parts, including finite element integration, detection of material contacts, and particle interaction if smoothed particle hydrodynamics is used to model highly deforming materials. The need to balance all of these computations simultaneously is a difficult challenge that has kept many commercial and government codes from being used effectively on parallel supercomputers with hundreds or thousands of processors. Our strategy is to load-balance each of the significant computations independently with whatever balancing technique is most appropriate. The chief benefit is that each computation can be scalably parallelized. The drawback is the data exchange between processors and extra coding that must be written to maintain multiple decompositions in a single code. We discuss these trade-offs and give performance results showing this strategy has led to a parallel implementation of a widely-used solid mechanics code that can now be run efficiently on thousands of processors of the Pentium-based Sandia/Intel TFLOPS machine. We illustrate with several examples the kinds of high-resolution, million-element models that can now be simulated routinely. We also look to the future and discuss what possibilities this new capability promises, as well as the new set of challenges it poses in material models, computational techniques, and computing infrastructure.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, make any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

1 Introduction

Solid dynamics simulations are among the most widely used engineering tools. Crash simulations, a prototypical example, consume more time on Cray vector supercomputers than any other industrial application [11]. Commercial and government-sponsored codes such as DYNA [24, 12], PamCrash [14, 15], and ABAQUS [1] can be used to answer safety and reliability questions in a variety of crash and impact scenarios of interest to industry and government. For example, will an automobile frame protect its occupants in a side-impact collision? Will a shipping container containing hazardous materials burst open during an accident? At what point will a bridge joint fail due to increased loading? If smoothed particle hydrodynamics are included in the finite element models, then problems with fluids or highly-deforming solids or fluid/structure interactions can also be studied.

In all of these cases, running models with sufficient resolution, long-enough timescales, and high-fidelity material models can be extremely computationally intensive. This makes the codes natural candidates for parallel implementation. Besides enabling current simulations to be run faster, an efficient parallel implementation would allow new and significantly larger problems to be addressed. More detailed geometries could be modeled, along with more complex constitutive models.

Despite these attractions, large-scale parallelization of solid mechanics calculations has proven to be quite difficult. A number of previous implementations have failed to scale beyond a few dozen processors. Reasons for this will be discussed in Section 2. A parallel crash code running efficiently on 16 or 32 processors is clearly valuable, but the increasing availability of larger parallel machines is motivation to develop more scalable techniques.

In this paper we describe our general strategy for parallelizing solid mechanics calculations. By using separate assignments of computational work to processors for different stages of the calculation, we have developed what we believe is the first truly scalable approach for running these codes effectively on massively parallel supercomputers. Over the past few years we have used the PRONTO-3D transient dynamics code as a test-bed for our strategy. It is similar in scope to the codes listed above and also includes an SPH modeling capability.

We sketch our parallelization strategy in Section 3. At the end of that section we provide performance results for PRONTO-3D running on the Sandia/Intel TFLOPS machine. In Section 4 we illustrate the kinds of analyses being performed with parallel PRONTO-3D. The ability to run at this scale has made a qualitative difference in the kinds of modeling questions that can be analyzed and answered. Finally, in Section 5 we discuss some of the lessons learned from our effort. In particular, we highlight some additional challenges and opportunities for both the computational mechanics and parallel computing communities in the arena of solid mechanics simulation.

2 Background

A finite-element (FE) solid mechanics simulation performs at least two significant computations each timestep. The first of these is the computation of stresses and strains within the FE mesh. The second is the detection of "contacts" between pairs of interpenetrating surface elements. These interactions are illustrated in Fig. 1. The basic idea is that no two bodies can occupy the same space at the same time. When contact pairs are detected, "push-back" forces are computed to correct the positions of the elements at the end of the timestep to prevent unphysical interpenetration of two solid objects.

If the simulation includes a gridless Lagrangian formulation, like smoothed particle hydrodynamics (SPH), then a third computational step is required. This is the calculation of the interactions of each SPH particle with its neighboring particles. Unlike, finite elements which have static connectivities (to nodes) to compute gradients, SPH particles must search for their nearest neighbors to compute the necessary gradients. This requires a global search for geometric neighbors inside a sphere of influence around each particle. SPH particles also need to be included in the contact detection and push-back computations.

Parallelization of finite element techniques and particle methods have been well studied in isolation. We will briefly highlight the approaches used in the next section. However, the contact detection task is unique to solid mechanics simulations and has proven to be a stumbling block to efficient parallelization of such codes on large numbers of processors. The reason for this is that the contact detection problem has three characteristics which make it difficult to load-balance: it is global, irregular, and dynamic.

In the parallel sense, contact detection requires global communication because any two processors can potentially own a pair of contacting surfaces. This is in contrast to the FE computation which only requires an exchange of information between elements connected by the topology of the FE mesh. Thus a processor need only communicate locally with a few neighboring processors to perform the FE computation. In contact detection, a surface element on one processor can impinge upon an element that earlier was far away and owned by any other processor.

For similar reasons, the communication required to detect contacts is irregularly structured. Processors need to exchange information if the mesh or particles they own are in close proximity to each other. Instead of simple, regular patterns like those supported in standard message passing libraries, this requires complex interactions between processors.

Contact detection is dynamic because which pairs of surfaces are close enough to each other to potentially interpenetrate changes dramatically over the course of a simulation. Again, this is in contrast to the FE computation which can take advantage of the mesh connectivity and resulting interprocessor communication pattern being static. In contact detection, some kind of dynamic communication pattern is necessary to continually acquire current information about which other surfaces are near a processor's contact surfaces.

Given these difficulties, how can we efficiently parallelize the task of contact detection? Several authors have reported on their efforts to parallelize contact detection on SIMD or data-parallel computers [18, 20, 25]. None of these efforts exhibited scalability on large numbers of processors. These results are consistent with the general observation that unstructured problems are difficult to parallelize effectively on SIMD machines. In part for this reason, SIMD architectures have largely been displaced in the high-performance computing world by message-passing and shared-memory parallel computers.

On message-passing and shared-memory machines the most common approach in transient dynamics simulations has been to use a single, static decomposition of the mesh to perform both FE computation and contact detection [14, 15, 16, 17, 19, 22]. At each timestep, the FE region owned by a processor is bounded with a box. Global communication is performed to exchange the bounding box's extent with all processors. Then each processor sends contact surface and node information to all processors with overlapping bounding boxes so that contact detection can be performed locally on each processor. Though simple in concept, this approach is problematic for three reasons. First, contact detection only involves the surface of the mesh, while the FE analysis also includes interior elements. Thus, a good decomposition of the FE problem may do a poor job of balancing contact detection. This is not as severe an issue for problems dominated by shell elements (e.g. [17]). Since every shell element is on a surface, a single decomposition can balance both parts of the computation. Second, the geometric extent of a processor's sub-domain may become large, so the bounding box surrounding that processor's elements will overlap with many other processor's boxes. Even if the initial overlap is small, as objects move and deform during the calculation large overlaps can ensue. Communicating potential contact information will then require large amounts of communication and force the processor to search a large fraction of the global domain for its contacts. Third, the bounding box enclosing a processor's elements will generally contain a significant amount of volume outside those elements. Consequently, much of the data communicated to a processor is unnecessary. This causes extra time to be spent in the communication and also when searching for contacts.

A different approach to parallelizing contact detection was developed by Har [8]. In this work, the 3-D geometry is divided into bins, and processors are dynamically assigned bins in which to search for contacts. The dynamic assignment is handled with a master/slave technique. All the information is sent to processor 0, who parcels it out to the other processors. This approach suffers from two significant scalability limitations. First, processor 0 must have enough memory to hold the entire contact problem. And second, communication costs are quite high to send all the data to processor 0 and then have processor 0 send it out again.

An alternative approach, more similar in spirit to our work and which was developed concurrently, is described by Hoover, et al. [12]. This approach uses a different decomposition for contact detection than for the FE analysis. In their method, they decompose the contact surfaces and nodes by overlaying a fine 1-D grid on the entire simulation domain and mapping the contact surfaces and nodes into the 1-D "slices". A variable number of slices are assigned to each processor so as to load-balance the number of contact elements per processor. Each processor is responsible for finding contacts within its collection of slices which it can accomplish by communicating with processors owning adjacent slices. While this approach is likely to perform better than a static decomposition, its 1-D nature limits its utility on large numbers of processors. The implementation described in [12] suffered from some load imbalance on as few as 32 processors of a Cray T3D.

In summary, to our knowledge no previous attempts at parallelizing contact detection have scaled to more than a few dozen processors.

3 Algorithmic Details

In this section we outline the algorithms and load-balancing techniques that can be used to parallelize a solid mechanics code. Full details of our work with PRONTO-3D can be found in some of the references [2, 3, 21]; here we simply highlight the general strategy. We believe these ideas are generally applicable to any solid mechanics code, as well as other multi-physics finite element codes, an issue we return to in Section 5.

As discussed in the previous section, a mechanics timestep consists of several stages. A typical sequence is listed in Fig. 2 for a dynamics code using an explicit integrator. A slightly adapted sequence of steps would be valid for a quasi-statics code using an implicit solve to compute, for example, the relaxed position of a gridded object subject to a load. These details would not change the basic concepts outlined in the discussion to follow.

The FE computation in step (1) of Fig. 2 uses the material properties of the physical elements to compute a local stress or force acting on each element due to its surrounding connected elements. In general, this applied force will deform the element and thus the physical object described by the Lagrangian mesh as the simulation proceeds. This FE computation will be load-balanced if each processor owns an equal number of mesh elements that require the same amount of time to process. For cases where different elements require different compute times due to material model differences, each element can be weighted proportional to its CPU cost, and elements assigned to processors so as to balance the total weight across processors. The communication cost within the FE step will be low if the elements owned by any one processor have minimal connection to elements owned by other processors. Because the mesh connectivity is essentially fixed for the duration of a simulation, static load-balancing techniques can be used to achieve these goals. We use the software package Chaco [10] as a pre-processor to perform the partitioning; it has several graph-based algorithmic options suitable for the task. Similar FE parallelization strategies have been used in other transient dynamics codes [8, 12, 14, 15, 17, 19]. In practice, parallel efficiencies of over 90% can be achieved with these methods when large meshes are mapped to thousands of processors.

In step (2), SPH computations are performed. The interested reader is referred to [4] for details of how SPH is formulated within a solid mechanics code. The key point is that two particles interact if their spheres of influence intersect. To implement this efficiently in parallel there are two requirements: that each processor own an equal number of particles, and that the region owned by a processor be geometrically compact so that neighbors can be found quickly and with a minimum of communication. An added difficulty is that during the course of a simulation, particle density can undergo large fluctuations – in a simulation of an explosion, for example. Thus no static spatial decomposition of particles to processors will satisfy these two requirements over time. Instead we use a dynamic load-balancing technique known as recursive coordinate bisectioning (RCB) [6, 13] for the SPH particles and re-balance them each timestep (or every few timesteps). RCB assigns a rectangular sub-domain containing equal numbers of particles to each processor. The simple rectangular shape of the sub-domain makes it easy to find neighboring particles in other processor's sub-domains. RCB has the additional advantage that small movements of the particles do not cause large perturbations in the decomposition, which means data movement is minimized when re-balancing occurs. The only additional communication cost in the SPH computation is the exchange of particle information (positions, forces, etc) across the *surface* of each processor's RCB sub-domain. Since the SPH computation itself scales as the *volume* of the sub-domain, the step (2) portion of the timestep scales well in parallel.

Step (3) of Fig. 2 is the time integration; forces previously computed on mesh elements and SPH particles are used to advance their positions and velocities. This step is perfectly parallel since each processor owns equal numbers of elements and particles and no interprocessor communication is required.

Step (4) is the contact detection task. The movement of mesh elements and particles in step (3) will cause unphysical interpenetrations like those depicted in Fig. 1. An efficient parallelization of this task is critical, since in a serial simulation it can require over 50% of the run time. For reasons outlined in the previous section, a static decomposition, such as the one used for assigning FE mesh cells to processors, will not balance the contact search. Rather, we need a dynamic decomposition of all objects that will potentially come in contact (surface elements and SPH particles). If each processor ends up with equal numbers of objects, then the search for contacts will be load-balanced. Additionally, since contact surfaces have a finite extent or bounding box that encompasses their volume of influence as they move during the timestep, we will need to acquire contact surfaces from neighboring processor's domains. Similar to the SPH discussion, this will have minimal cost if each processor's sub-domain is geometrically compact and shaped to make this acquisition fast. In fact, the RCB method used for SPH particles again satisfies all these requirements. Thus at the beginning of step (4) all the contact surfaces (faces of elements on the surface of objects) and SPH particles are balanced across processors via RCB, using an earlier decomposition as a starting point. Note that although we use the same routine, this is a different decomposition on a different set of data than the SPH RCB decomposition.

When the RCB contact decomposition has been created, each processor owns a rectangular box containing equal numbers of contact objects. Contact surfaces with bounding boxes that extend beyond a processor's sub-domain are cloned and sent to neighboring processors. Each processor is now ready to search for contacts within its box. A key point is that this search is conceptually identical to the global detection problem we originally formulated, namely to find all the contacts between a collection of surfaces and nodes bounded by the entire simulation domain. In fact, at this point in step (4), each processor simply calls the original unmodified serial contact detection routine. This is important for computational speed and also for maintainability since the serial contact-detection algorithm may have various optimizations to efficiently find contact pairs. In PRONTO-3D, for example, the serial detection routine spatially sorts the contact nodes in multiple dimensions, then searches the sorted lists for each surface's bounding box to find possible contacting nodes [9]. This list of possible contacting nodes is then processed to determine those actually in contact and to calculate push back directions and magnitudes. All of this computation can now take place on-processor without communication. The resulting information about contact pairs is communicated back to the FE and SPH decompositions to prepare for the next step.

In step (5) of Fig. 2, each processor computes push-back forces on the small number of its particles or mesh elements that came into contact on the current timestep. These new forces are used in step (6) to adjust the positions of the interpenetrating elements and particles.

In summary, we have outlined a parallelization strategy for a prototypical solid mechanics simulation that uses 3 different decompositions within a single timestep: a static FE-decomposition of mesh elements, a dynamic SPH-decomposition of SPH particles, and a dynamic contact-decomposition of contact surfaces and SPH particles. We use a graph-based decomposition for the mesh elements; the other two decompositions are updated by RCB. The key advantage of this approach is that each of the 3 critical time-consuming stages of the timestep is load-balanced independently. The disadvantage is that extra communication must be performed to move mesh and particle information between the decompositions and to update the dynamic decompositions. However, if this cost is not too high, we expect the strategy to scale to large numbers of processors.

As evidence of the scalability of this approach, we present two sets of timing data for parallel PRONTO-3D running on the Sandia/Intel TFLOPS machine at Sandia National Labs. This machine has 4500 computational nodes, each of which has two commodity 200 Mhz Pentium-Pro processors and 128 Mbytes of memory [23]. The nodes are configured in a 2-D mesh with a proprietary communication network that supports a 350 Mbyte message-passing bandwidth between nodes (in the limit of long messages) with 20-30 microsecond latencies. The implementation of the parallel algorithms described above and of PRONTO-3D itself is all within a distributed-memory message-passing paradigm using standard F77 and C code with MPI library calls for interprocessor communication.

Our first test problem is the crushing of an idealized shipping container by an inclined wall.

Both the wall and the cylinder are modeled as perfectly plastic materials with properties of steel. This is a pure FE problem; no SPH particles are included. It is a stringent test of the contact detection algorithm's robustness, since the container crumples on itself as it is crushed nearly flat, similar to a soda can being stepped on. Fig. 3 shows CPU timings for a series of scaled-size crush simulations where the container and wall are meshed more finely as more processors are used. On one processor a 1875-element model was run. Each time the processor count was doubled, the number of finite elements was also doubled by refining the mesh in a given dimension. Thus the leftmost data points are for a 3750 element simulation running on 2 processors; the rightmost points are for a 6.57 million element simulation on 3504 nodes of the TFLOPS machine. The topmost curve in Fig. 3 is the total CPU time per timestep averaged over a 100 microsecond (physical time) run. On the small problems this is a few hundred timesteps; on the large problems it is several thousand, since the timestep size must shrink as the mesh is refined. The lower curve is the portion of time spent in the FE computation. Contact detection is the time between the lower and middle curves, about one-half of the total CPU time for this simulation. Additional overhead, including the contact push-back, is the time between the top two curves.

The timing data shows excellent scalability of the parallel algorithms on thousands of processors. Linear speed-up would be horizontal lines on this plot; the FE computation scales nearly perfectly. Contact detection consumes a roughly constant fraction of the time for all runs. The variations in these timings are primarily due to the changing surface-to-volume ratios of mesh elements as refinement is done in different dimensions. The total CPU time begins to rise in a non-scalable way on the largest $P = 2048$ and $P = 3504$ runs because the normally small push-back computation becomes somewhat unbalanced on very large numbers of processors.

A second set of timing results are for a parallel PRONTO-3D simulation of a metal penetrator impacting a target at approximately 700 ft/sec. These were simulations performed by Kurt Metzinger at Sandia. P:ctures of similar simulations are shown in the next section. The complex geometry of the penetrator was modeled with 155,000 finite elements; the target was modeled with 415,000 SPH particles since it undergoes large deformations during the impact. Timing data for this fixed-size problem running on the TFLOPS machine is shown in Table 1 on varying numbers of processors. Average per-timestep timings are shown for the FE computation, the SPH computation, and the contact detection which in this case includes both mesh-mesh and mesh-particle contacts.

Due to memory limitations the smallest number of nodes this problem could be run on was 64. The last column in the table indicates the speed-up relative to the 64-processor run as a baseline. The timing data shows that PRONTO-3D achieves good parallel performance even for fixed-size hybrid FE/SPH problems where all 3 decompositions are active every timestep.

4 Applications

In this section we highlight several computational studies currently being performed with the parallel PRONTO-3D code by colleagues at Sandia.

4.1 Airplane Crash

Crash simulations are the most common kind of simulation performed using nonlinear transient dynamics codes. The level of detail that can be modeled is limited primarily by the size of the model. Sandia has been asked to study the safety of igloo structures at a facility used to store nuclear weapon components.

Determining whether a storage structure can withstand the impact of an incoming airplane is difficult. As the aircraft impacts the igloo, fracturing of the fuselage and wings occur. The analysis is further complicated by the need to model the fuel leaking from the wing tanks and potentially causing a fire or explosion. The goal in this project was an order-of-magnitude prediction of the impact loads on the structure and of the fuel dispersion as opposed to a detailed analysis of the aircraft break up. We modeled the KC-135R aircraft with traditional finite elements and its fuel with SPH particles. While there are many different approaches (element free galerkin, j integral, crack tip opening angle, node release, etc.) one could use to handle the tearing and breakup of a thin skinned structure, the approach we have considered is element deletion. In this approach, an element is deleted when a parameter such as damage exceeds a critical value. Newly exposed surfaces are then added to the list of potential contact surfaces.

Initial simulations of this scenario concerned the fuel dispersal pattern from a wing impacting a vertical pole. Fig. 4 shows the fuel cloud and damage to the wing. In the image on the right the fuel particles are deleted to show the tearing of the wing more clearly. This calculation was run on 128 nodes of the TFLOPS computer using 110,000 hexahedral and shell elements and 130,000 SPH particles.

Current efforts are concentrated on modeling the entire aircraft. The full model is shown in Fig. 5 which uses 516,000 shell elements to model the airplane and 540,000 SPH particles to model the fuel. In the lower portion of the figure, the fuselage and fuel are removed to show interior detail. Fig. 6 is a simulation of the fuselage impacting an idealized rigid target. The fracturing in the nose tip is clearly visible. The next stage of the project is to model the ground with SPH particles using a volumetrically yielding soil model. This will bring the total model size to over 2 million elements. In the final analysis, the fuel, soil, airplane and igloo will all interact.

4.2 Foam Crush

A second application is the response of foams of various types. Foams are often used to distribute impact forces or to absorb energy in collisions. The macroscopic properties of foams depend upon their fine-scale structure in a complex way that is not well understood. Better constitutive models of foam properties can be obtained through simulation of small-scale behavior. Unfortunately, very large simulations are necessary to be able to compare computations to experiments.

The top portion of Fig. 7 depicts the initial state of a simulation of an open-cell foam, with cells about one mm in diameter. A linear elastic material model was used for the strut material, but the complex buckling and folding generates complex nonlinear behavior. The foam is crushed from above by a fast moving plate while resting on a rigid surface. The lower portion of the figure reveals there is some crush near the impacting plate, but the majority of the compaction occurs on the opposing boundary. This is due to the reflection of stress waves off the bottom plate and is consistent with experimental observations.

Each of the foam struts was modeled with multiple hexahedral elements, for a total of about 900,000 elements. Previously, beam finite elements were used, but the complex deformation patterns associated with large crush were difficult to capture with beam elements. By contrast, hexahedral elements are able to model very complex contact conditions. The drawback to using hexahedrals, aside from the number of elements needed, is that a very small timestep is required to properly integrate the motion. This problem was run on 512 nodes of the TFLOPS machine and required 8.8 hours of CPU time to run 650,000 timesteps. The complexity of the model and the contact interactions between struts can be appreciated in the close-up view shown in Fig. 8. Red regions in this figure are those sustaining highest stress.

4.3 Metal Forming

A third application is the simulation of metal forming, a widely used manufacturing process for thin metal sheets. In this process, a metal plate is squeezed between two dies to produce the desired shape, such as an automobile body panel. In many cases the final shape of the formed piece is dependent not only on the shape of the dies but also on the friction between the dies and the workpiece. The most accurate way to model metal forming is to use hexahedral elements that can capture the local thinning or thickening of the plate and the associated normal forces that determine the friction response. The simulations are often performed using quasi-statics codes to ensure convergence at every increment as the metal is molded to the die. Although the details of contact enforcement are different in a quasi-statics code, the contact detection operation is essentially identical to that performed in a transient dynamic code such as PRONTO-3D. In fact we have ported our parallel contact detection modules into JAS3D, a quasi-statics code derived from PRONTO-3D. This has allowed us to simulate this class of problems on massively parallel machines.

Fig. 9 shows a calculation run by Gerry Wellman. A screen is pressed between two dies with the screen held in place by blank holders. One of the questions that can be studied by simulation is the effect of die orientations to see how precisely they must be manufactured. If a moderate amount of imperfection in the dies does not dramatically reduce the quality of the final screen, the savings could be significant. The initial analysis shown in the figure modeled only a 10 degree arc of the screen. A side view including the blank holders with both dies is shown at the top of the figure. The 10 degree wedge contains about 45,000 hexahedral elements and its forming can be simulated quickly on 32 processors. The bottom portion of the figure shows the final formed screen pressed against the die. The full model will include at least a 180 degree section and possibly a full 360-degree circular screen depending on the orientation of the dies. This will require about 1.6 million elements to model accurately, with contact detection and push-back consuming a large portion of the computation time.

4.4 Projectile Impact

A final application is that of penetration problems where a projectile traveling at high velocity impacts and pierces a target. This class of problem is very challenging to model for several reasons. First, due to the large deformations that occur in the target, traditional Lagrangian meshes often suffer from element inversion. One approach that avoids this problem is to model the target with SPH techniques. Second, the penetrator structure can be quite complicated and we are often interested in the response of its interior to the collision. This requires a large number of elements to attain sufficient resolution.

An example calculation due to Kurt Metzinger compared a coupled FE-SPH simulation to experiments for the oblique penetration of a steel rod through an aluminum plate at a velocity of 400 m/sec. Fig. 10 shows excellent agreement between x-ray photographs of the experiment and the simulation results. Initially, both sets of data show the penetrator bending upwards. As it perforates the plate, it turns over and bends downward. As the resolution of the meshed penetrator is increased, the degree of bending in the final shape approaches that of the experiment.

A more sophisticated computation is shown in Fig. 11 for an impact at 200 m/sec. The internal geometry of the penetrator is modeled with 155,000 hexahedral elements with full contacts. The target is modeled with 415,000 SPH particles. An outer ring consisting of 6,500 4-node shell elements is used to constrain the target material from free radial expansion. A full 3-D model is necessary for this simulation because the penetrator has a 3-D geometry and oblique impacts are important. This calculation was run for 161,000 timesteps on 1024 nodes of the TFLOPS machine and required about 36 hours to complete. The ejecta of material at both the entrance and exit of the target is consistent with experiments for this type of event. The lower portion of the figure shows a cut-away view with the hole that has been cut through the target.

5 Vistas

As shown in the previous sections, adapting state-of-the-art solid mechanics codes and algorithms so they execute efficiently on massively parallel supercomputers is a challenge requiring the combined skills of engineers, finite element analysts, and computational and computer scientists. This effort has resulted in a solid mechanics code which can run scalably on thousands of processors. In this section we discuss what the payoffs are for this effort as well as what other issues are blocking further progress.

One obvious benefit of a code being able to run on thousands of processors is that a new class of simulations can be attempted for the first time. Questions can be asked and addressed at length and time scales that could not be contemplated before. However, even running PRONTO-3D on hundreds of processors has meant a enormous leap in the kinds of problems that are accessible to the analyst on an every-day basis with reasonable turn-around time. Consider that a node on the TFLOPS machine consisting of two commodity Pentium CPUs is now nearly as fast as a single processor of a Cray Jedi. machine running a version of PRONTO-3D optimized for that platform. Thus even a few hundred processors can perform simulations many times faster than they could have been performed previously on traditional vector processors. This power can be used to increase the spatial or temporal resolution of a simulation grid, to run a more accurate material model, to do parametric studies of a system's response under varying conditions, or to embed the dynamics simulation inside an optimization loop where parameters are varied to optimize the design of a critical component. Alternatively, stochastic sampling can be used to gain a clearer understanding of the range of possible responses, instead of performing a single, deterministic calculation.

However, the increase in compute power does not always have the impact one would anticipate. Consider that for a 3-D model, an increase in spatial resolution by a only a factor of two, along with the requisite halving of the timestep when grid cell sizes are reduced, requires a 16-fold boost in compute power. Also, the natural tendency to run bigger and longer calculations brings a new set of challenges, even if the simulation itself can be run quickly. Finite element grid generation is often extremely difficult for multi-million element models of complex geometries. The data sets generated by the dynamics can easily consume 100s to 1000s of Mbytes. Even if the parallel machine can dump these data sets quickly to parallel disk farms, they typically must be reassembled for post-processing analysis or moved to another large-memory platform for visualization. And we have found current visualization tools are generally inadequate to deal with data sets of these sizes. All in all, the analyst often spends considerably more time on the pre- and post-processing phases of the problem than waiting for the simulation to run.

Given these bottlenecks it is natural to ask if there is a need for solid mechanics (or many other) codes to scale to hundreds or thousands of processors. Since the majority of current parallel computing platforms are more modest in size, aren't algorithms that enable a code to run reasonably well on 16 or 32 processors sufficient? Aside from the need to occasionally run huge problems, which can only be done on large numbers of processors, we offer two answers to this question.

While predicting the future of computing platforms is difficult at best, and is driven by forces far removed from scientific computing, one current trend is clear. There is a cost advantage to building a parallel system out of commodity parts – CPUs, memory, networks, disks, etc. Distributed-shared memory (DSM) machines such as the SGI Origin and similar offerings from other vendors who sell high-end integrated systems are popular purchases for those who can afford them. However there is a growing grassroots movement to build-your-own parallel computer. Beowulf-style clusters of Pentium or DEC Alpha chips, strapped together with fast Ethernet or Myrinet networking, are springing up in laboratories and academic departments at an astonishing rate. The communication bandwidths and latencies on clustered systems are poor compared to DSM machines or even traditional distributed-memory machines like the Intel Paragon or TFLOPS or Cray T3E. This places a premium on minimizing the communication cost of an algorithm if it is to run well on even a few dozen processors. We also note that for the code developer, the ideal scenario is to write code that will run on any platform. The message-passing programming paradigm, implemented with portable MPI library calls, is a robust solution. All parallel machines of the foreseeable future, be they distributed or shared memory, will support MPI. Coding in this style forces the application developer to think about how to store data locally on the processor which will perform the computation on those data. This is generally advantageous even on shared-memory machines that support global memory access. In short, our experience has been that implementing the most-scalable, minimum-communication, distributed-memory algorithm, is often a win in the long term, even if it means more effort up-front, because the code will run efficiently on any platform.

A second justification for our development effort is that we have discovered that our algorithms and parallel strategy can be re-used in a variety of other codes and applications. For example, our contact-detection algorithm has been ported into three other Sandia codes: JAS3D - a derivative of PRONTO-3D used for quasi-statics problems like the metal forming example in the previous section, ALEGRA - a new mixed Lagrangian/Eulerian shock physics code, and SIERRA - a new multi-physics code being developed as a framework for runing a variety of coupled mechanics, thermal, and fluid-flow simulations. Since some of these codes are works-in-progress, it is difficult to predict how sensitive their parallel performance will be to load-balance issues, even on a few dozen processors. Undoubtably, their parallel performance will be problem dependent. Thus we feel safest having designed algorithms that should perform well in whatever application they become a part of.

At this point, it is worth saying a few words about the low-level design of our parallel algorithms. We have found it advantageous to write our tools with an object-oriented design in mind. This can be done in standard C (or even F77/F90) with a bit of forethought without the performance sacrifice that object oriented languages often impose. For us the chief benefit has been an easier path to re-use of certain functionality in other, sometimes unforeseen, applications.

A brief example illustrates this point. The SIERRA framework mentioned above is designed to solve problems where an object may be overlaid with multiple grids – e.g. one for mechanical deformation, another for thermal conduction. Within a timestep there is the need to interpolate solutions back and forth from one grid to the other. The interpolation is complicated by the fact that the grids may adapt or move independently during the course of a simulation. Computationally this *grid transfer* task entails the following: For each nodal point of the first grid, find the element in the second grid which contains it. Then interpolate the solution quantity (or quantities) of interest to the nodal point using the nodal values of the containing element. In parallel, this is more complex because there is no guarantee the two grids will be decomposed to processors in the same way, particularly if they adapt or move over time.

As we developed a parallel algorithm for the grid transfer operation, we discovered we were addressing many of the same issues that arose in parallel contact detection. How do you collect dispersed data so that a single processor can do a local search computation? How do you find which processors own neighboring sub-domains so that elements which extend beyond one processor's domain can be shared appropriately? And what is the optimal way to setup communication patterns between irregular groupings of processors? The answers to these questions can use many of the same load-balancing and low-level communication operations we implemented for contact detection. Embedding these functions in a high-level grid transfer operation was made easier by defining a clean interface to the functions and hiding many of the data structures they generate from the calling program by storing them in the code modules themselves. These are object-oriented design principles that have been successfully used in other scientific libraries as well [5, 7]. The result is a toolkit of load-balancing and communication primitives that can be assembled in interesting ways to perform grid transfer or contact detection or other parallel load-balancing and decomposition tasks.

Finally, what are the near-term challenges for the solid mechanics research community? As is often the case, having faster codes serves to amplify other weaknesses in one's problem-solving methodology. We have already indicated some bottlenecks in the pre- and post-processing phases that need to be addressed by the computational science community. However, the "brute-force" solution of adding millions of elements to solve a hard mechanics problem may not always ensure accurate solutions to the physics of the problem. Two alternative approaches to increasing accuracy are better material models and adaptive gridding technology.

The constitutive relations implemented in a material model embody the central physics underlying any solid mechanics simulation. PRONTO-3D is used to simulate the response of an enormous variety of materials – from concrete, metal and wood to ceramics, gasoline and soil. As grid sizes shrink, there is also a growing need to model composite materials, i.e. those with heterogeneous composition at a small length scale. With the resolution made possible by parallel mechanics codes, the fidelity of simulations may now be limited by the material models. Conveniently, in some cases, the ability to model multi-million element grids can help in the development of better material models. For example, consider the foam simulations of the previous section. For materials that have a microstructure, a detailed simulation can be performed at the micro-scale. The results can be used to validate a continuum-level constitutive relation for the material that can be used in other, more macro-scale simulations.

On-the-fly grid adaptation in response to stress gradients induced in a physical object offers a potentially great savings in the number of finite elements needed to solve a problem to a desired accuracy. More commonly used in fluids calculations, adaptive methods are an active area of research in solid mechanics simulations. Issues of what criteria should drive the adaptivity, how to insure the new meshes conform to the deforming objects, how to best interpolate solutions from old to new meshes, are all open questions. On a parallel machine additional issues arise, such as how to generate the new mesh consistently across processor boundaries, or how to dynamically load-balance if the new mesh is not distributed evenly among the processors. Achieving adaptivity within a multi-physics code using multiple grids, all running efficiently on a large parallel machine, is a grand-challenge level goal that will require continued long-range effort from the solid mechanics community.

6 Acknowledgements

We thank our many collaborators at Sandia National Laboratories who have worked with us to design these algorithms and implement them in PRONTO-3D, including David Gardner, Courtenay Vaughan, Jeff Swegle, and Martin Heinstein. We also thank John Pott for his calculations on the aircraft impact, Mike Neilsen and Steve Attaway for their work on the foam analysis, Gerry Wellman for his work on sheet metal forming problem and Kurt Metzinger for his work on the penetration problems. This work was performed at Sandia which is operated for the Department of Energy under contract No. DE–AL04–94AL8500. It was partially supported under the Joint DOD/DOE Munitions Technology Development Program, and sponsored by the Office of Munitions of the Secretary of Defense.

References

[1] *Abaqus/explicit theory manual*, tech. rep., Hibbit, Karlsson & Sornesen, Inc, 1080 Main Street, Pawtucket, RI 02860-4847.

[2] S. Attaway, T. Barragy, K. Brown, D. Gardner, B. Hendrickson, S. Plimpton, and C. Vaughan, *Transient solid dynamics simulations on the Sandia/Intel Teraflop computer*, in Proc. SC97, ACM/IEEE, November 1997.

[3] S. Attaway, B. H. S. Plimpton, D. Gardner, C. Vaughan, K. Brown, M. Heinstein, and J. Peery, *A parallel contact detection algorithm for transient solid dynamics simulations using Pronto3D*, Computational Mechanics, (1998). To appear.

[4] S. W. Attaway, M. W. Heinstein, and J. W. Swegle, *Coupling of smooth particle hydrodynamics with the finite element method*, Nuclear Eng. Design, 150 (1994), pp. 199–205.

[5] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, *Efficient management of parallelism in object-oriented numerical software libraries*, in Modern Software Tools in Scientific Computing, E. Arge, A. M. Bruaset, and H. P. Langtangen, eds., Birkhauser Press, 1997, pp. 163–202.

[6] M. J. Berger and S. H. Bokhari, *A partitioning strategy for nonuniform problems on multiprocessors*, IEEE Trans. Computers, C-36 (1987), pp. 570–580.

[7] M. Frigo and S. G. Johnson, *FFTW*.
WWW address: `http://theory.lcs.mit.edu/ fftw/`.

[8] J. A. Har, *New Scalable Parallel Fine Element Approach for Contact Impact Problems*, PhD thesis, Georgia Institute of Technology, Atlanta, GA, 1998.

[9] M. W. Heinstein, S. W. Attaway, F. J. Mello, and J. W. Swegle, *A general-purpose contact detection algorithm for nonlinear structural analysis codes*, Tech. Rep. SAND92–2141, Sandia National Laboratories, Albuquerque, NM, 1993.

[10] B. Hendrickson and R. Leland, *The Chaco user's guide: Version 2.0*, Tech. Rep. SAND94–2692, Sandia National Labs, Albuquerque, NM, June 1995.

[11] M. Heroux, Cray Research/SGI, 1997. Personal communication.

[12] C. G. Hoover, A. J. DeGroot, J. D. Maltby, and R. D. Procassini, *ParaDyn: DYNA3D for massively parallel computers*, October 1995. Presentation at Tri–Laboratory Engineering Conference on Computational Modeling.

[13] M. Jones and P. Plassman, *Computational results for parallel unstructured mesh computations*, Computing Systems in Engineering, 5 (1994), pp. 297–309.

[14] G. Lonsdale, J. Clinckemaillie, S. Vlachoutsis, and J. Dubois, *Communication requirements in parallel crashworthiness simulation*, in Proc. HPCN'94, Lecture Notes in Computer Science 796, Springer, 1994, pp. 55–61.

[15] G. Lonsdale, B. Elsner, J. Clinckemaillie, S. Vlachoutsis, F. de Bruyne, and M. Holzner, *Experiences with industrial crashworthiness simulation using the portable, message-passing PAM-CRASH code*, in Proc. HPCN'95, Lecture Notes in Computer Science 919, Springer, 1995, pp. 856–862.

[16] J. G. Malone, *Parallel nonlinear dynamic finite element analysis of three-dimensional shell structures*, Computers and Structures, 35 (1990), pp. 523–539.

[17] J. G. Malone and N. L. Johnson, *A parallel finite element contact/impact algorithm for non-linear explicit transient analysis: Part II – parallel implementation*, Intl. J. Num. Methods Eng., 37 (1994), pp. 591–603.

[18] R. R. Namburu and D. A. Turner, *Contact impact algorithm on a data parallel computer*, in High-Performance Computing in Computational Dynamics, ASME, 1994, pp. 111–119.

[19] P. Persson, *Parallel Numerical Procedures for the Solution of Contact Impact Problems*, PhD thesis, Linköping University, Linköping, Sweden, 1996.

[20] E. J. Plaskacz, T. Belytschko, and H. Y. Chiang, *Contact impact simulations on massively parallel simd computers*, Computing Systems in Eng., 3 (1992), pp. 347–355.

[21] S. Plimpton, S. Attaway, B. Hendrickson, J. Swegle, C. Vaughan, and D. Gardner, *Transient dynamics simulations: Parallel algorithms for contact detection and smoothed particle hydrodynamics*, J. Parallel Distrib. Comput., (1998). To appear.

[22] Q. Stout, University of Michigan, 1998. Personal communication.

[23] WWW address: `http://mephisto.ca.sandia.gov/TFLOP/sc96/index.html`.

[24] R. G. WHIRLEY AND B. E. ENGEMANN, *Dyna3d, a nonlinear, explicit, three-dimensional finite element code for solid and structural mechanics - user manual*, Tech. Rep. UCRL-MA-107254 Rev. 1, November 1993.

[25] Z. H. ZHONG AND L. NILSON, *Contact impact algorithms on parallel computers*, Nuclear Eng. & Design, 150 (1994), pp. 253–263.

Figure 1. *Various types of mesh/mesh and particle/mesh contacts. (a) Self contact. (b) Two distinct bodies penetrating one another. (c) Two interpenetrating surfaces. Enforcement must occur at nodes B and F. (d) Particle/mesh contact.*

(1) Compute FE forces on each mesh element
(2) Compute SPH forces on each particle
(3) Move mesh elements and particles
(4) Detect mesh/mesh and particle/mesh contacts
(5) Generate push-back forces
(6) Final update of mesh and particle positions

Figure 2. *One timestep of a solid mechanics calculation.*

Figure 3. *Average CPU time per timestep on the TFLOPS machine to crush a container meshed at varying resolutions. The mesh size is 1875 finite elements per processor at every data point. The lower curve is finite element computation time; the middle curve includes both FE and contact detection time; the upper curve is total CPU time including contact push-back.*

Table 1. *Breakdown of CPU seconds per timestep to perform a penetrator simulation on different numbers of processors P of the TFLOPS machine.*

Figure 4. *Combination finite element and smoothed particle hydrodynamics simulation of fuel dispersal from an airplane wing.*

Figure 5. *Two views of a finite element grid representation of an aircraft used for crash scenario simulations.*

Figure 6. *An aircraft fuselage nose impacting a rigid target.*

Figure 7. *Finite element representation of a foam between two plates before and after compression.*

Figure 8. *Close-up view of the foam struts after compression. The colors represent local stress with red indicating regions of high stress.*

Figure 9. *Side and top views of a metal screen (pink) about to be pressed between two dies (blue and green). After forming, the screen has molded to the die configuration.*

Figure 10. *Comparison of experimental X-ray (left) and simulation results (right) for penetration of a steel rod through an aluminum target.*

Figure 11. *Simulation of a penetrator impacting a target. The penetrator is modeled with finite elements; the target is modeled with SPH particles.*

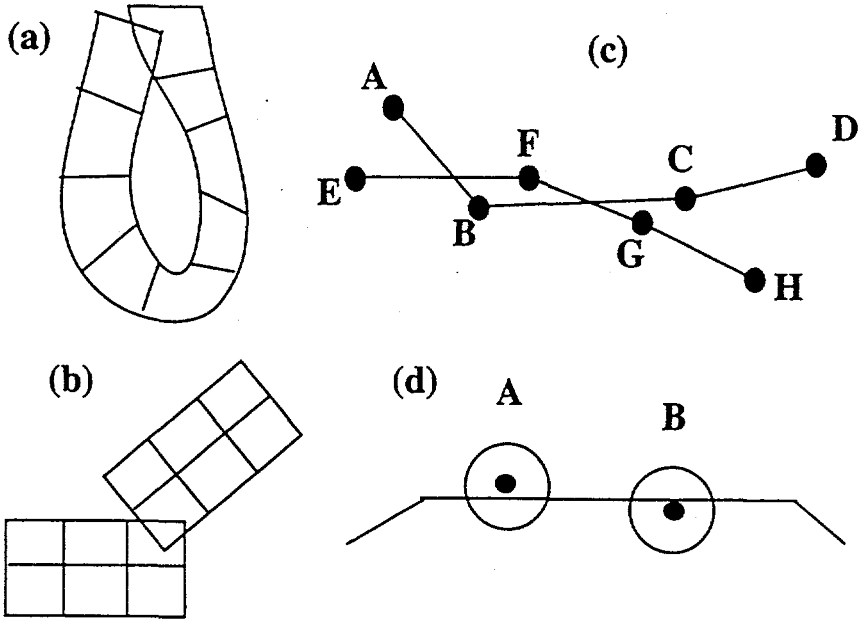

Fig 1

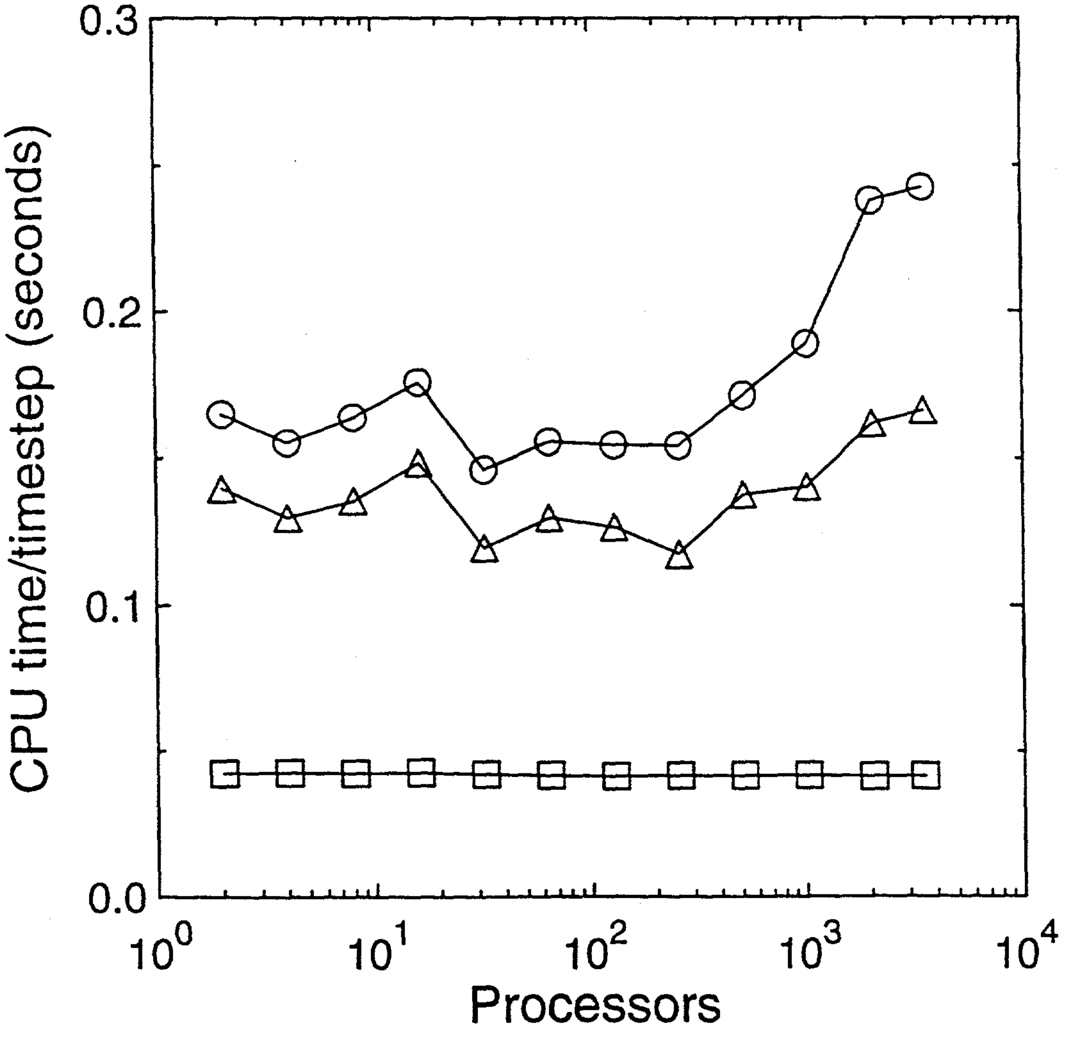
0.3
0.2
0.1
0.0
CPU time/timestep (seconds)
10^0
10^1
10^2
10^3
10^4
Processors

Fig 3

P	FE	SPH	Contacts	Total	Speed-up
64	0.409	4.55	1.24	6.27	1.00
128	0.191	2.36	0.715	3.30	1.90
256	0.102	1.30	0.533	1.97	3.19
512	0.0525	0.708	0.334	1.11	5.65
1024	0.0303	0.423	0.238	0.704	8.90

Table 1

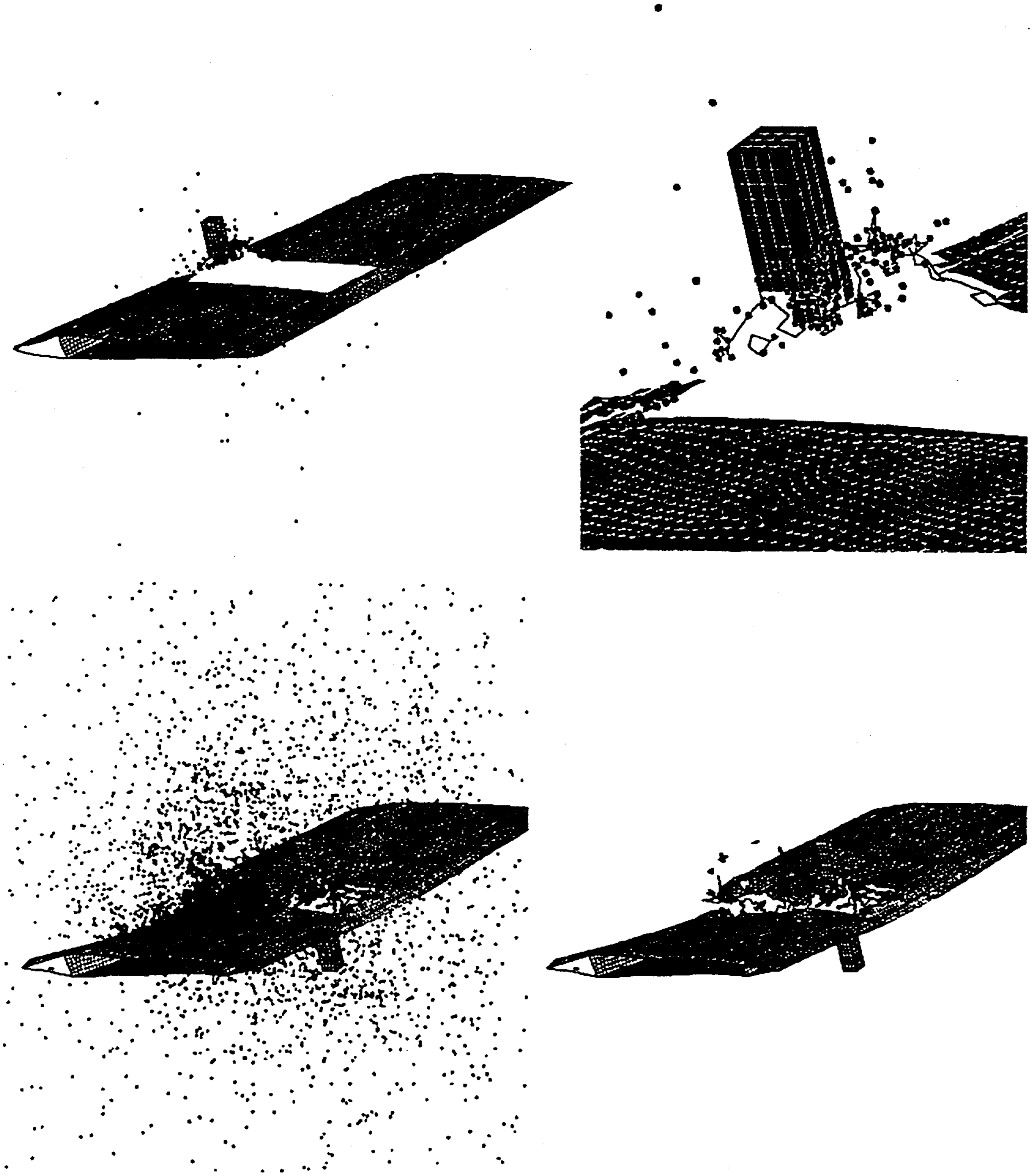

Fig 4

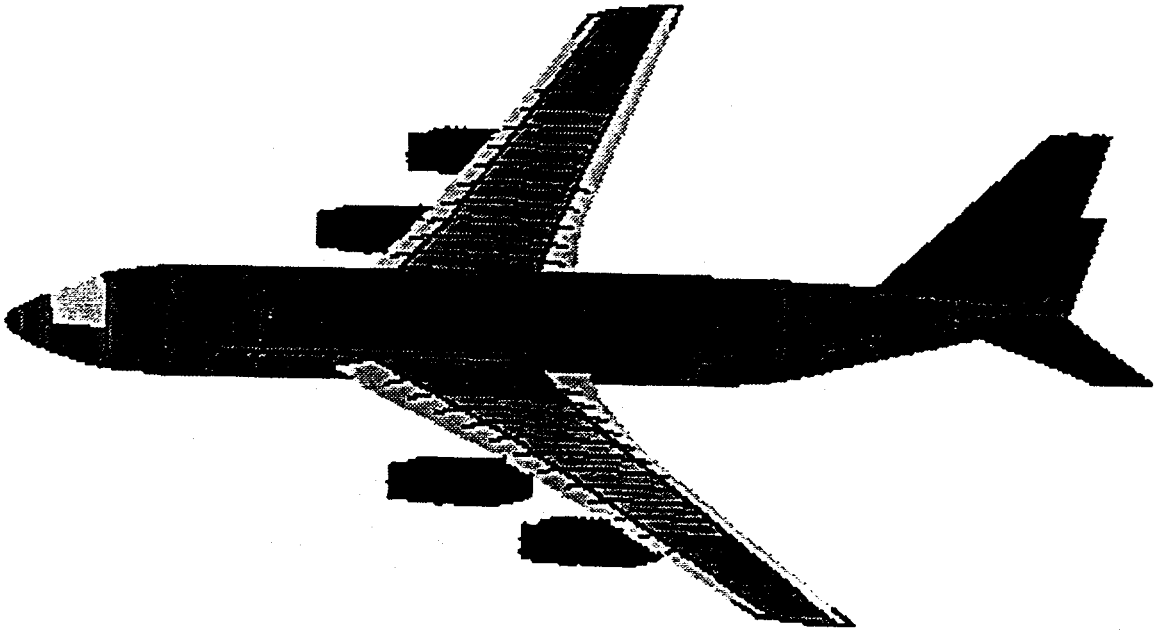

Fig 5a

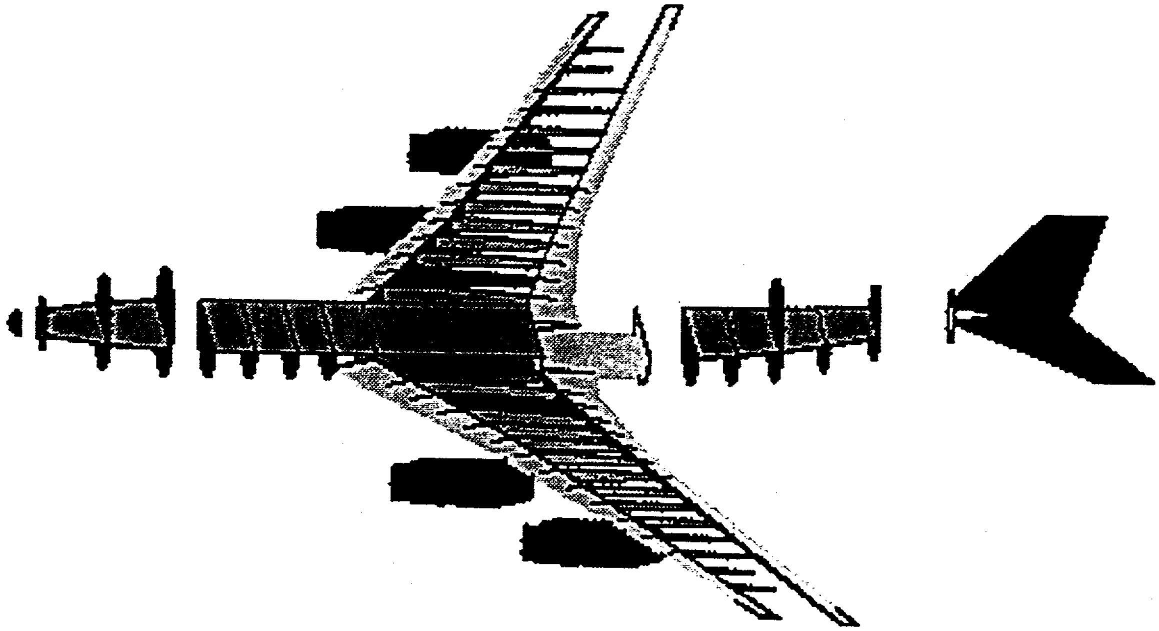

Fig 5b

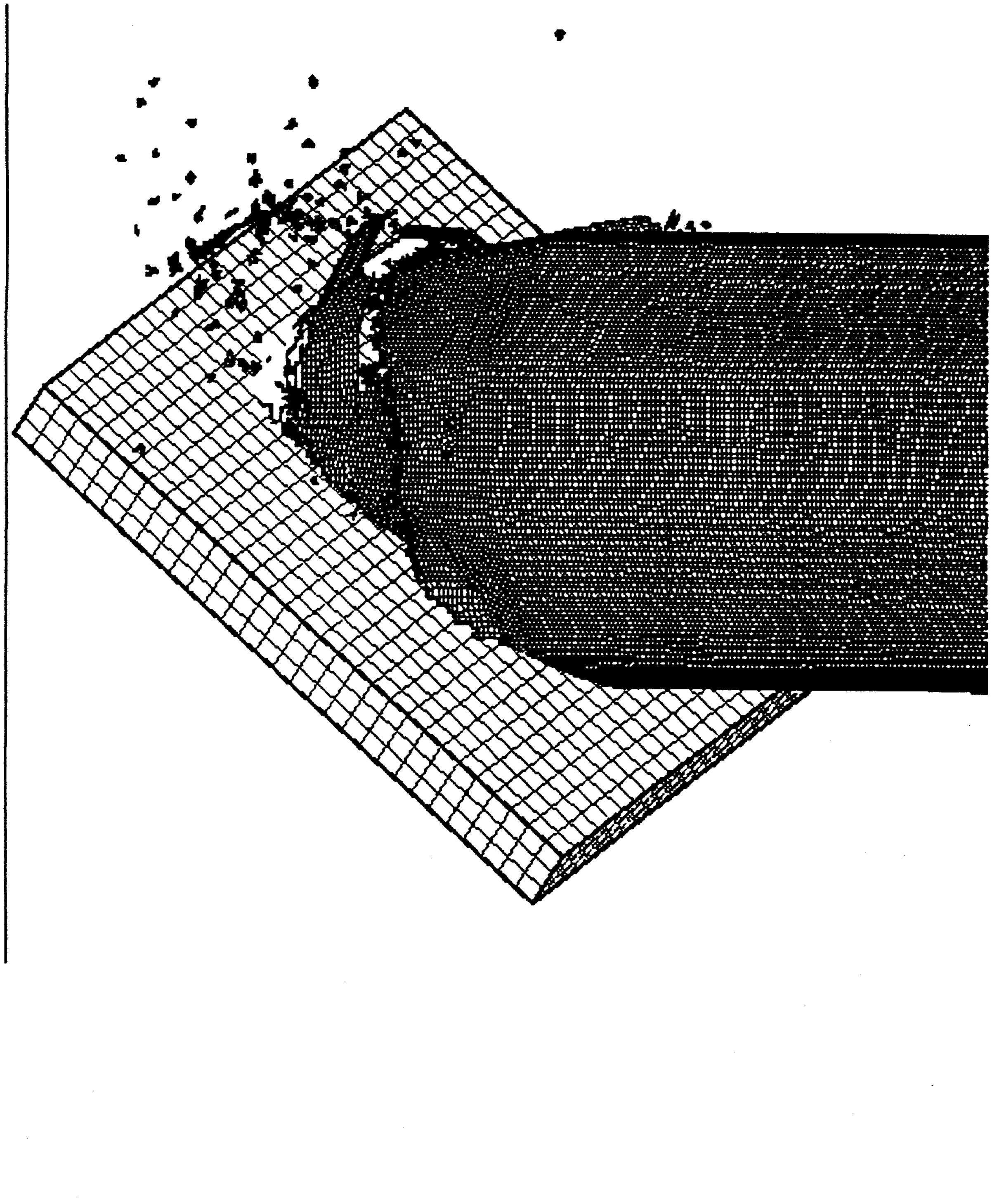

Fig 6

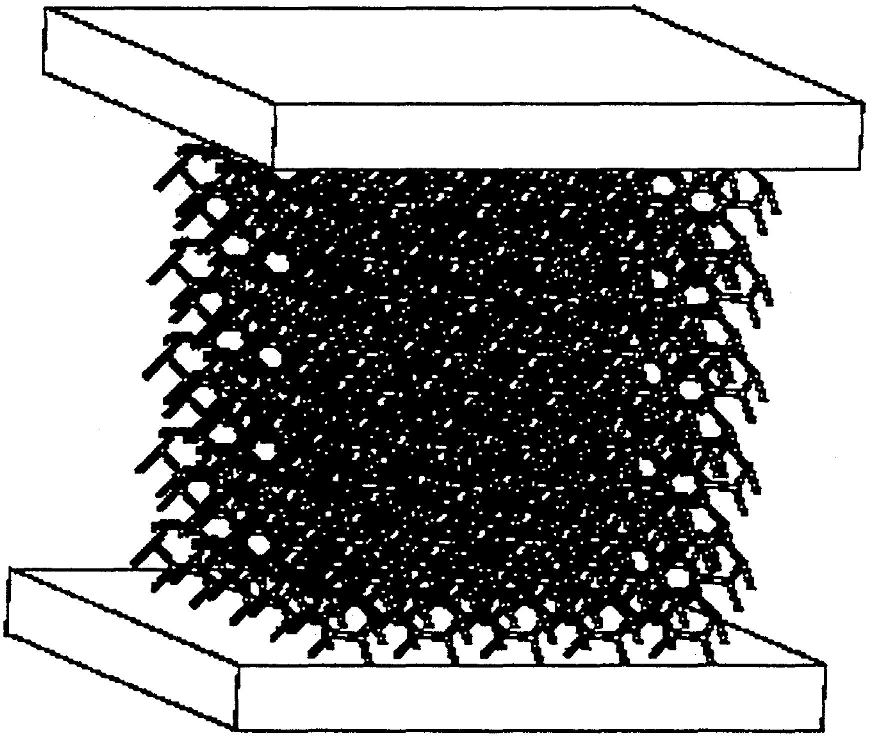

Fig 7a

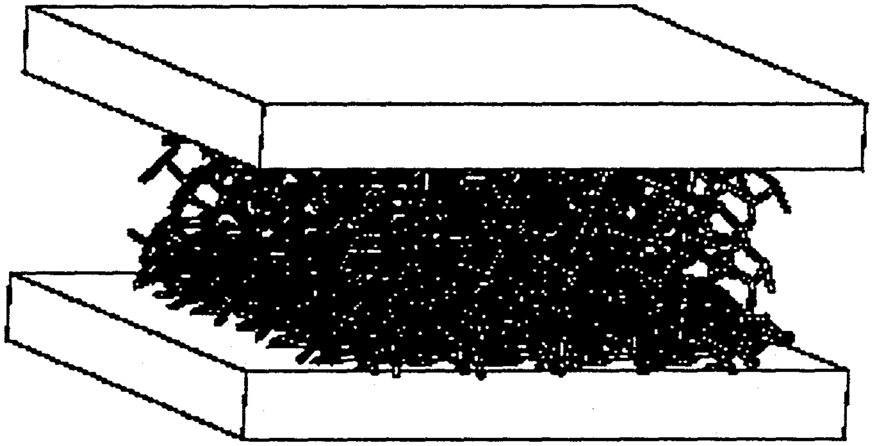

Fig 7b

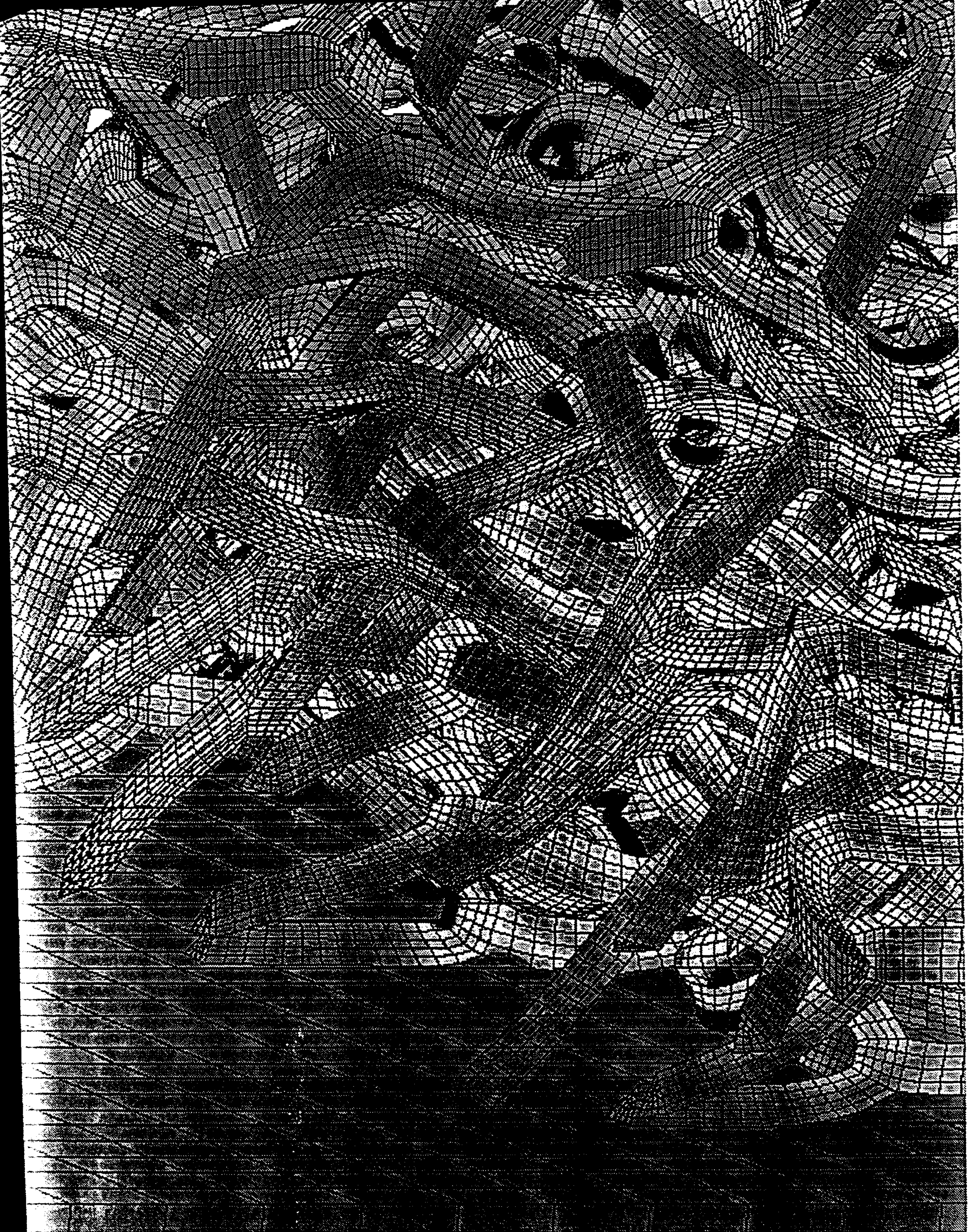

Fig 5

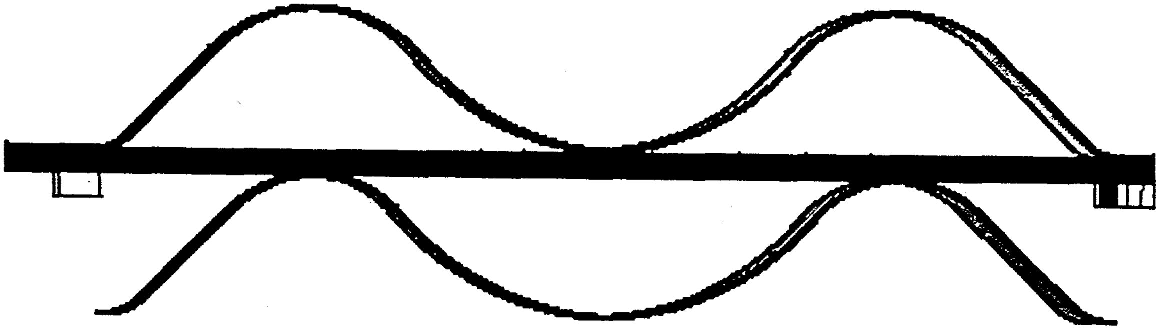

Fig 9a

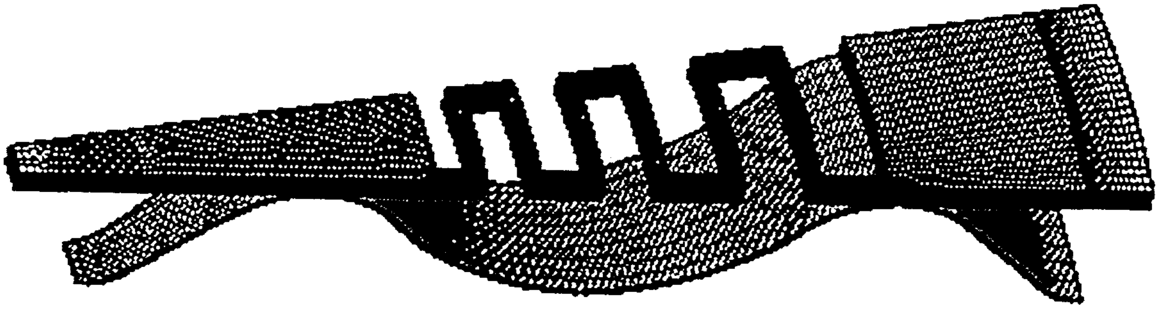

Fig 9b

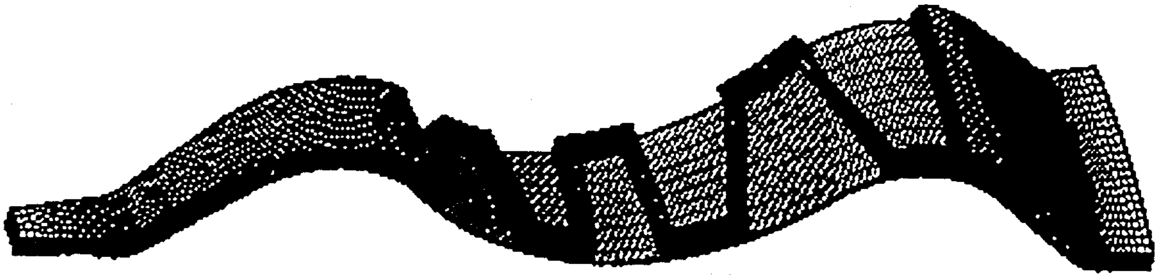

Fig 9c

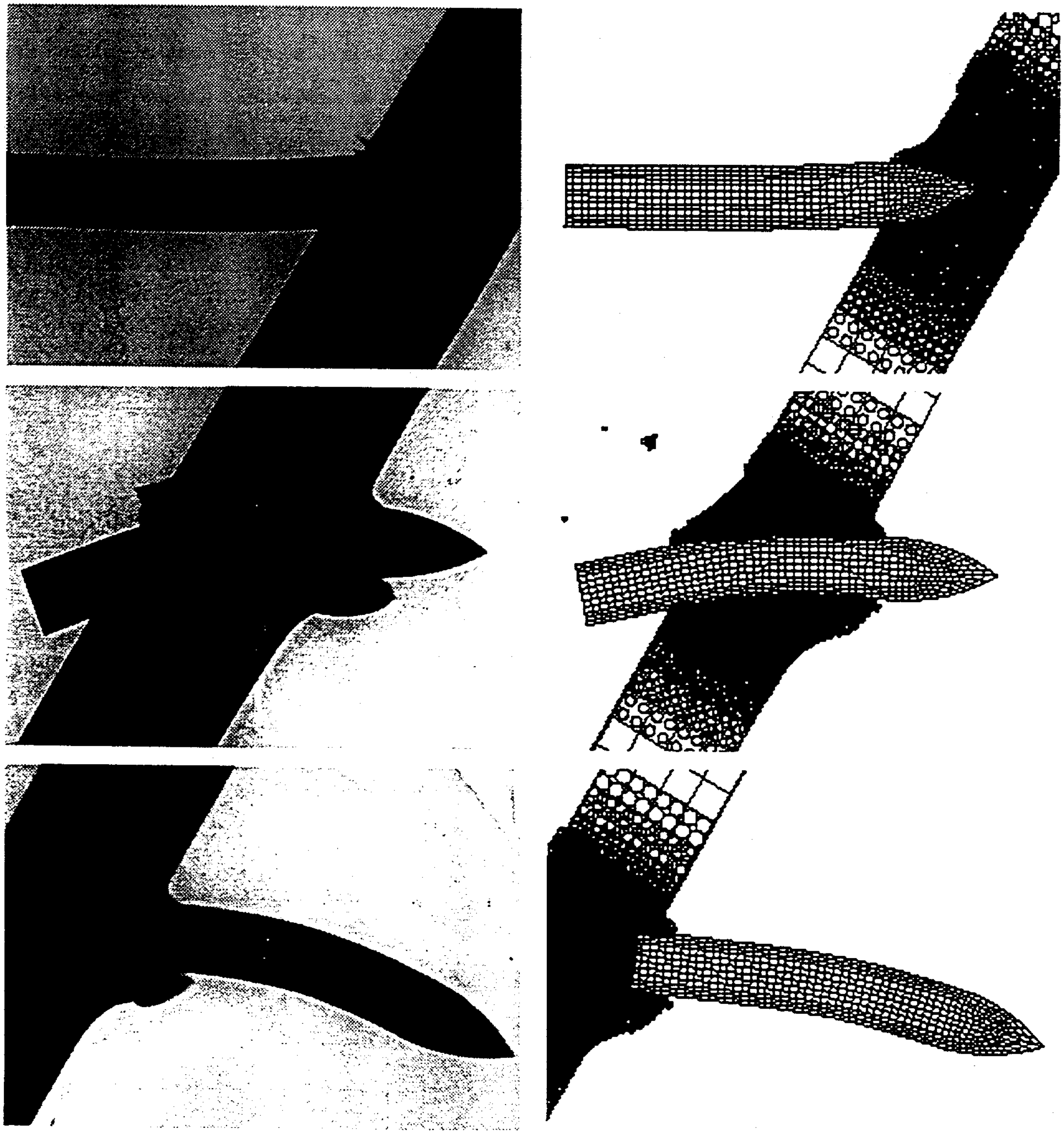

Fig 10

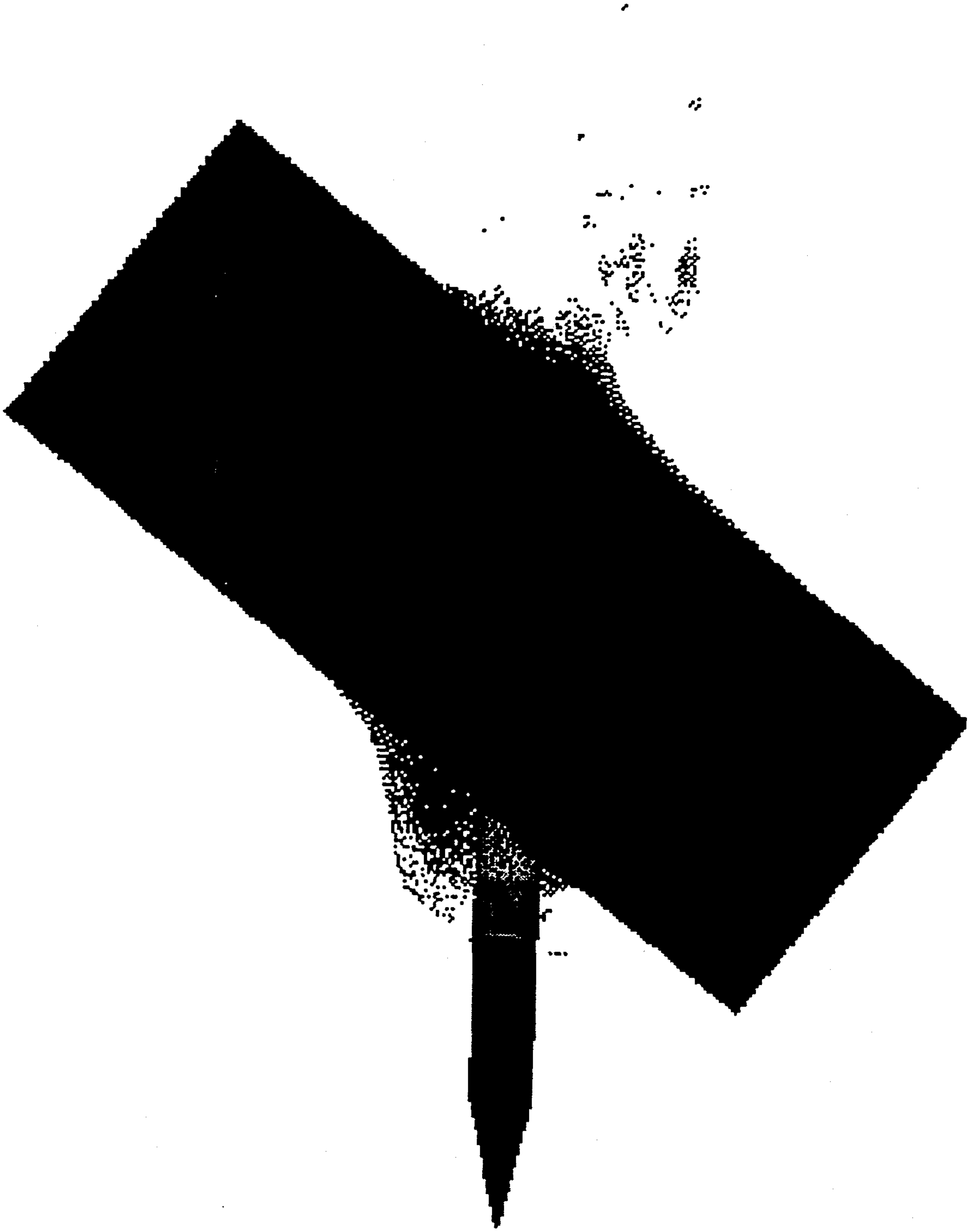

Fig 11a

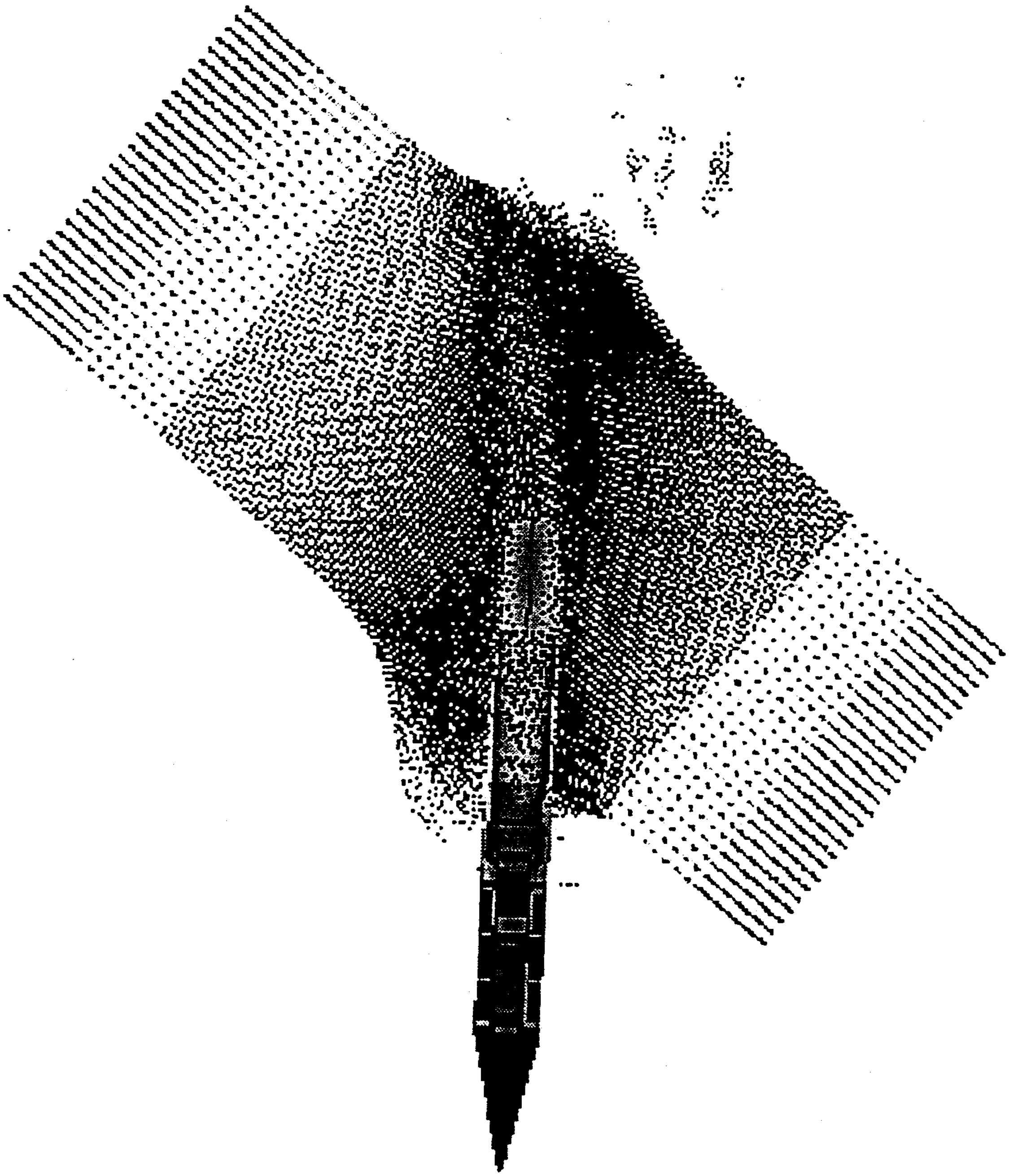

Fig 11b