Chapter 7

Implementation Issues

In this chapter we review issues relevant to the implementation of dynamically coupled particle systems. Of importance are the issues of efficiently computing the forces for large numbers of particles, computing the set of nearby neighbors for each particle, the stable and efficient numerical integration of the computed forces over time, and methods of interactively visualizing the system.

7.1 Efficient Force Computations

The state of a dynamically coupled particle system at a given time is described by the set of particle positions, orientations, translational and angular velocities. Using this state information, the forces and torques on each particle can be computed directly from the inter-particle force functions. By numerically computing a discrete approximation of the force over the time interval, the system state can be updated to the next time step.

The tractability of these computations is a limiting factor in the ability to interact with these systems in real-time. With appropriate assumptions we can reduce the complexity in both time and memory requirements. In this section we discuss the computational problems, possible solutions, and our choices.

7.1.1 Direct Pairwise Computation

The definition of the force on a particle i due to the other particles, is the sum of the pairwise inter-particle forces

$$\mathbf{f}_i = \sum_{j \neq i}^{N} \mathbf{f}_{ij},$$

where N is the number of particles. A straight forward computation of the forces for the above equation is conceptually and computationally simple, but not necessarily the most efficient. Such an evaluation will require $O(N^2)$ operations, making it reasonable for small systems, yet prohibitively expensive for large values of N. Since we may use on the order of several thousand particles we need a better solution.

7.1.2 Mesh Methods

When individual inter-particle forces are small, yet the cumulative effects of inter-particle forces are significant, then a mesh method may be useful in reducing the amount of computation (Hockney and Eastwood, 1988). If the forces can be described in terms of a continuous potential field, then a mesh approach is valid. The basis of the approach is to approximate a field that is continuous in space by a set of discrete values on a finite mesh covering of the space.

The total cost of the mesh approach is proportional to both the number of particles and the number of mesh points. Thus, for a coarse grid, the mesh method may result in large gains of speed over the direct approach. One drawback is that the gain in speed is traded for a loss of accuracy, as each step in the process introduces new errors. In addition, the accuracy is highly dependent upon the relationship between the continuous functions and the mesh spacing. In order for the mesh to adequately approximate a system, the mesh spacing must be smaller than the important wavelengths of the system. That is, potential fields are poorly represented on the mesh for distances less than the mesh spacing. Mesh techniques become useful when the problem involves smoothly varying forces.

To approximate the Lennard-Jones force would require a mesh spacing significantly smaller than the average inter-particle spacing, resulting in more mesh points than particles. Thus, for our problem, the mesh approach will consume more time than the direct $O(N^2)$ approach.

7.1.3 Combined Methods

The third approach combines features of the direct and mesh approaches, taking advantage of their respective strengths (Hockney and Eastwood, 1988; Greengard, 1988). It is adequate for *highly correlated systems* with smoothly varying long-range forces. The key of the approach is to split the inter-particle forces into two parts:

$$\mathbf{f} = \mathbf{f}_s + \mathbf{f}_l \tag{7.1}$$

a rapidly varying short range force $\mathbf{f}_s$ which is nonzero for only a few inter-particle distances, and the slowly varying long range force $\mathbf{f}_l$ which is sufficiently smooth to be accurately represented on a mesh. The direct method is used to find the total short-range force contribution on each particle, while the mesh method is used to find the total long-range contribution. The total cost is proportional to the number of particles and the total number of mesh points. This is a reasonable approach for the potential energies we are using.

7.1.4 Limited Force Range

The final approach is to compute the short range forces using the direct particle-particle computation, and to ignore distant forces. This is a valid assumption in highly correlated systems where there exists strong short range inter-particle forces and the distant forces are insignificant. Assuming a reasonably distributed set of particles, there will be at most a small number of neighboring particles that contribute to the total force of a given particle, resulting in $O(N)$ computation time for the entire system of N particles.

The drawback of this approach is that it is restricted to forces that decay to near zero at the force cut-off boundary, otherwise discontinuities will be introduced into the continuous force function. It is wise to avoid such discontinuities as they will create instabilities when we numerically integrate the forces.

7.1.5 Discussion

From considering the choices available to us, the combined method and the direct method[1] with a limited force range are the two computationally feasible approaches. The mesh method is not suitable for highly correlated systems such as ours, and the full direct method would require $O(N^2)$ operations per time step, thereby limiting interactive shape modeling to small systems of particles.

The nature of our applications suggests the use of the direct method with limited force range as the best solution. We have designed our inter-particle potential functions so that the forces decay as a function of distance, allowing distant forces to be ignored. Still the potential discontinuity at the boundary introduces a force discontinuity contributing to numerical instabilities. To alleviate this we can use the weighting function (3.12) which goes to zero at the boundary, thus insuring a continuous force function. In addition to computational efficiency considerations, limiting the potential functions provides a consistent potential energy representation of the merging and splitting characteristics of our dynamically coupled particle systems. When two objects are separated by the force range distance, the potential energy description of the two objects is also independent.

An important consideration, we have neglected so far in our analysis, is the computational cost required to find the neighbors of each particle. We discuss this next.

7.2 Neighbor Computation

We now look at the problem of finding the set of neighboring particles for each particle. This problem is formalized as the range search problem.

7.2.1 The Range Search Problem

An essential condition of computing the inter-particle forces is finding the neighbors of each particle. This can be stated as finding an $O(N)$ subset of particle pairs from the $O(N^2)$ possible pairs, such that each selected pair is no farther apart than a specified distance r. This type of search problem, known as the *range search* problem, has been studied extensively in the computer science literature (Overmars, 1983; Preparata and

[1] The names "direct", the "mesh", and "combined" methods also go by the names of "particle-particle", "particle-mesh", and "particle-particle-particle-mesh (P^3M)" respectively(Hockney and Eastwood, 1988).

Shamos, 1985; Samet, 1989; de Berg et al., 1997; Goodman and O'Rourke, 1997). Formally we state this as:

> Given a set of three-dimensional points $\{x_1, x_2, \ldots, x_N\}$, and a sphere of radius r, centered at position $\mathbf{x}_s$, which points $\mathbf{x}_i$ satisfy the condition $\|\mathbf{x}_s - \mathbf{x}_i\| \leq r$, where $\|\mathbf{x}_s - \mathbf{x}_i\|$ is the Euclidean distance measure?

A single range query is easily performed in linear time, by examining each of the N points. Linear space will suffice as only the positions must be stored. However, to compute the force of the system, we will need to find the nearby neighbors of each particle, resulting in $O(N^2)$ operations. In addition to the computing forces, the range search problem appears in triangulation, particle creation heuristics, and in the use of modeling tools. Similar to our desire to reduce the force complexity to something tractable, we want to reduce the computational complexity of the range search problem.

The approach to choose depends upon the application at hand and properties of the data, such as: Are the range queries of a constant size search radius or variable search radius? Is the data set fixed, or will points be added and deleted? Are the data points uniformly distributed throughout space, or highly localized?

We consider the approach to use in light of the application areas of this thesis: computer assisted animation, geometric modeling, and surface reconstruction. Below we list the qualities of the various applications, while noting that the needs of specific applications within these fields may vary. For all applications, the range queries will generally be of a small range for the force, triangulation, and particle creation heuristics. Likewise we can expect the use of modeling tools to range from including all of the particles to only a few. For all applications the points will be uniformly distributed in space at the local level, though non-uniform at the global level. Volumes will have denser local grouping of particles than will surfaces. For all application areas we can expect cases when we will want to add or delete particles from the system. The basic search operation required, in order of frequency, are: (1) range search, (2) insertion, (3) deletion, and (4) point existence. The last three operations assist in interactively grabbing, moving, adding, and deleting particles.

In the remainder of this section, we review several types of spatial data structures and comment on them in relation to our dynamically coupled particle system. Because our points sets and queries will be relatively well behaved, we will focus our discussion on simple data structures that will perform well on average, instead of focusing on complex data structures optimized for worst case performance. We begin with the hashing, followed by direct access grid structures, and end with multi-resolution tree structures.

7.2.2 Hashing

Hashing is a process used to quickly distribute and retrieve data among a fixed number of memory locations. Given a data point $\mathbf{p}_i$, a hash function $H(\mathbf{p}_i) = A_i$ can be used to compute an address A_i which is the address of a block of reserved memory, called a *bucket*, in which the data point is stored for later reference. Frequently this is implemented using low level bit operations to provide fast index computation. It has become common usage to define all indexing functions which use low level bit operations as hash functions. An example of such a function[2] is given in Graphic Gems (Glassner, 1993)[page 343]. Instead, we will use the theoretical definition of hashing as given below.

The theoretical goal of hashing is to take highly structured and sparse data in the domain, and provide a mapping into a uniformly distributed range, thus reducing the amount of memory required while providing constant time access. This is normally accomplished by defining the hash function to be a randomizing function with uniform distribution. A given input to the hash function must always produce the same output address in order to store and subsequently retrieve data values. Thus the function cannot be truly random, but rather pseudo-random. A random and uniform distribution of data, has the advantage of reducing the amount of memory required to be of the same order as the size of the data set. A second advantage is the hash function provides direct access to each data item for efficient insertion, retrieval, and deletion. There are, however, two corresponding disadvantages (Samet, 1989). First, it is difficult to find a suitable hash function that will map each data point to a unique location. Second, since hashing functions randomize the data, the function destroys the spatial relationships between data points. Destruction of the spatial relationships makes such a hashing scheme unsuitable for solving the range search problem.

7.2.3 Fixed Grid

The fixed grid is a spatial data structure that partitions space into equal constant size cells, with each cell storing the points falling within the cell volume. By partitioning 3D space along orthogonal planes, the mapping from a bounded R^3 volume to the finite number of cells can be directly computed from each point's x, y, and z values. The extent of the 3D grid can be easily computed by finding the minimum and maximum values of x, y, and z values. Provided the point set is uniformly distributed, the fixed grid can provide constant time insertion, deletion, point existence, and range search queries of small search radii. Thus it is an appealing solution to our problem.

The performance of the fixed grid depends on the ratio of the grid volume search to the actual search volume, the bucket size, and the distribution of points in space. As the cell width decreases, a smaller volume of the grid is searched, but at the cost of accessing more cells. A standard cell width to choose is one equal to the search radius. This requires testing the cell in which the center of the search radius falls, and the adjoining cells, for 27 cells in all. Using half the cell width results in accessing five times as many cells and approximately half the total volume. Using double the cell width results in accessing only eight cells, but almost three times the total volume. The most efficient search will result in a tradeoff between accessing more cells and testing fewer points. The optimal balance between the cell width and search volume depends on the relative costs of the accessing more cells and of testing a point for

[2]This function concatenates the high order bits of three integer indices together. The result is a new integer index.

inclusion. Both are implementation dependent. The number of points in an average cell is also dependent on the distribution of particles. Due to the short range repulsive and long range attractive forces, particles will be uniformly distributed at a local level, in sheets for surfaces and in tightly packed clusters for volumes. At the global level, the shape of the object determines the distribution. In general, we suggest using a cell width equal to the search radius.

The amount of memory required to represent the grid is dependent upon the number of cells. The number of cells is a function of the extent of the grid and the cell width. We begin by considering a simple example, such as modeling a sphere. If we model the sphere by volumetric particles, the number of cells in the grid will $O(N)$ where N is number of particles. If we model the sphere by surface particles, the number of cells in the grid will be $O(N^{3/2})$. This indicates that for large systems of oriented particles the majority of the grid will not be used. A 2D example of this is shown in Figure 7.1(a). Of course, we could construct cases where the solid model is sparse in space, such as a long thin twisted wire. In this case the grid will grow faster than linear in the number of volumetric particles. Likewise, we could construct cases where a surface has numerous folds effectively filling space. In this case the number of grid cells grows linearly with the number of surface particles. From a practical perspective we will assume dense solid models and surfaces that do not tend to fill space. Thus we will state that *in general* the amount of memory used will be of $O(N)$ for volume particles and $O(N^{3/2})$ for surface particles.

The final consideration is the bucket size. When the bucket size is too large, memory resources will be wasted. When the bucket size is too small, bucket overflows will occur requiring additional computation. To find a balance we consider the distribution of the points in space. Let us consider the two simple cases: (1) when particles are modeling a volume, and (2) when the particles are modeling a surface.

Volume Case

In the volume case, the particles are uniformly distributed throughout the volume of the object model. Due to the Lennard-Jones force, at equilibrium the particles will be separated by the equilibrium separation r_o, and under external forces we will assume no closer than the collision distance σ. To find an upper bound on the number of particles in a cell we will assume the cell is completely enclosed by the model volume, with particles represented by tightly packed spheres of diameter σ. From the volume packing density (Section 3.4.1) we can compute the number of tightly packed particles that will be present in a given volume, thus resolving the issue of bucket size. The upper bound on the number of particles in a cell is the integer ceiling of

$$P_v \frac{V_1}{V_2} = \sqrt{2}\left(\frac{w}{\sigma}\right)^3$$

where P_v is the volume packing factor for tightly packed spheres (3.20), V_1 is the volume of a cubic cell of width w, and V_2 is the volume of a sphere of diameter σ.

Surface Case

We now consider the uniform distribution of particles over the surface of a model. We could consider the case of a planar section cutting through the cell, but to create a better upper estimate we consider a surface of high curvature. Let us assume the cell encloses a hemisphere with diameter equal to the cell width. From the area packing density, we compute the number of particles in the cell as the integer ceiling of

$$P_a \frac{A_1}{A_2} = \frac{\pi}{\sqrt{3}} \left(\frac{w}{\sigma}\right)^2$$

where P_a is the area packing density for tightly packed circles (4.1.9), A_1 is the area of the hemisphere of diameter w, and A_2 is the area of a circle with diameter σ.

Summary

In summary, the grid provides fast constant time insertion, deletion, point existence, and range search queries for our problem when the optimal cell and bucket sizes are chosen as described above. We suggest a cell width equal to the search radius for efficient range searching. When modeling surfaces, in general, the number of grid cells will grow $O(N^{1/2})$ faster than the number of particles, suggesting this is not the best structure for surfaces. However when modeling volumes, in general, the grid grows linearly with the number of particles making it a reasonable choice.

7.2.4 Reduced Grid

For certain classes of data, we can combine aspects of the fixed grid data structure and hashing to create a "reduced grid" which limits the amount of memory required while maintaining fast storage, retrieval, and range searching (Szeliski, 1998). For this discussion we relax our definitions of the fixed grid and of hashing. We use the definition of hashing as the ability to quickly compute an index value and instead of randomizing the data, the hashing function will maintain the spatial structure inherent in the data set. The definition of a grid is modified so that cells map to multiple disjoint regions of the volume.

One reduces the number of cells in a grid of size M^3 by a factor of D^3 by applying a modulo based hashing function when computing the grid indices for each point. If the fixed grid indices for a point are the integers i, j, and k, then the reduced grid indices for this point would be $i \bmod E$, $j \bmod E$, and $k \bmod E$, where $E = M/D$. When E is a power of two, this is equivalent to masking out the high order bits of the index. One can also think of this as a redistribution of points through a modulo transformation of coordinate space. The result of this transformation is being able to reduce the number of cells necessary for storage, while keeping the original spatially close points in adjacent cells, where adjacency includes the concept of "wrapping around" the grid. A range search on the reduced grid will be guaranteed to return spatially close points, yet it may also returned distant points. A second drawback, is that point sets which are aligned with the major axes or point sets with

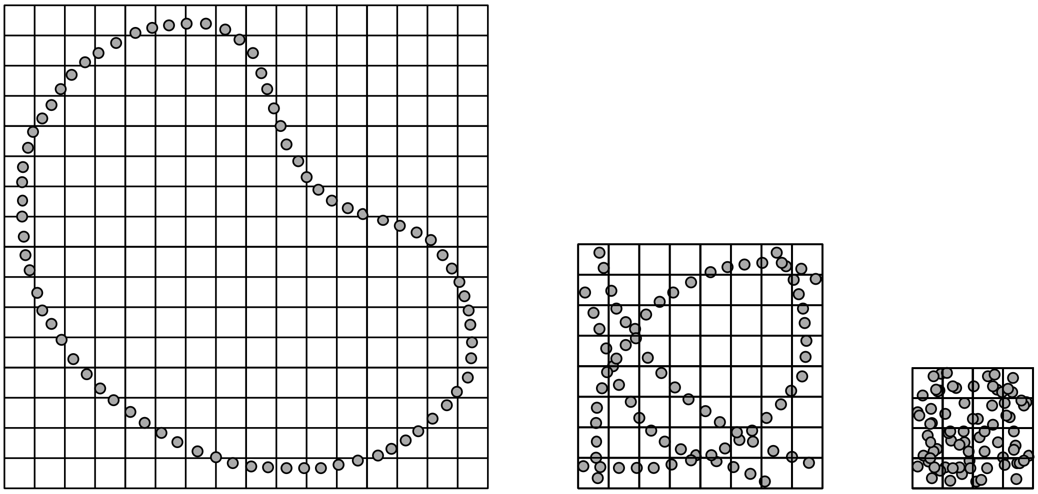


Figure 7.1: Storage of points in a grid.

(a) A 2D example a string of points stored in a 16x16 fixed grid. (b,c) Same points stored in reduced grid using modulo based indexing. (b) Grid reduced to 8x8, (c) Grid reduced to 4x4.

periodic structures may create high density regions in the reduced grid. Despite these drawbacks, reducing memory requirements by several orders of magnitude may be a sufficient reason for its use in certain applications. The reduced grid is illustrated in Figure (7.1) with a hypothetical 2D curve formed by a string of points.

7.2.5 Hierarchical Structures

One way to reduce the total number of unused cells and avoid the the bucket overflow problem is to adaptively vary the cell size through recursive subdivision (Samet, 1989). On overflow the volume of a cell is divided among a set of smaller cells, each with its own bucket. The original cell becomes a *parent* cell, and the new cells are *children* of the parent. As a child cell's bucket becomes full, it will become a parent cell, split, and create children. The relationship between the parents and children can be described by a directed tree structure where the nodes of the graph correspond to cells and the arcs between nodes correspond to the parent-child relationships.

Region-Based Tree Structures

A *region quad-tree*[3] is a directed tree-based data structure which recursively decomposes space based on the density of points. Initially, a quad-tree starts as a single

[3] In two dimensions, a quad-tree subdivides space into four quadrants. A three dimensional quad-tree, commonly called a oct-tree, subdivides space into eight octants. The quad-tree easily generalizes to higher dimensions. For generality we will use the term quad-tree to refer to such a tree in any dimension.

cell represented by the root of the tree. As data points are added, they are stored in the root cell's bucket until it is full. When full, the cell subdivides its space among 2^d equal size children cells each with its own bucket (where d is the dimension of the space), and then transfers each of its data points to the appropriate child. Applied recursively, this creates a tree structure.

Similar to the region quad-tree, the *region kd-tree* recursively subdivides space. At each level of the tree it subdivides space into two equal size regions, alternating the axis along which space is divided. This gives the kd-tree a branching factor of two at each node, compared to 2^d for the quad-tree.

The region quad-tree and region kd-tree exhibit identical theoretical time complexity and are suitable for our purposes. Building a tree of N points will take on average $O(N \log N)$ operations and require $O(N)$ memory. Point insertion, point deletion, and point existence queries each require $O(\log N)$ operations on average, where $\log N$ is the average distance from the root node to a leaf node. A range search can be performed in $O(F + N^{1-1/d})$ in the worst case, and in practice the query will take $O(F + \log N)$, where F is the number of points returned and d is the dimension. Since we expect particles to be evenly distributed over an area, at the local level, it is unlikely that the worst case time will actually occur.

Point-Based Trees

Point-based trees are spatial data structures used to encode a set of point data, by recursively subdividing space based on the location of individual points. Each node in the tree corresponds to a unique point in the data set. For a data set with N points, this results in exactly N nodes in the tree.

According to Overmars (1983), such quad-trees were the first data structures devised for solving the range search problem efficiently. In a *point quad-tree*[4] one of the data points is taken to be the root, and based on the value of the data point, a d-dimensional space is subdivided into 2^d quadrants, thus splitting the data set into 2^d subsets. These subsets, minus the original data point, will be encoded in the subtrees of the root node. This process is recursively applied until each quadrant contains at most one data point.

The *point kd-tree* was introduced as an improvement over the point quad-tree. While the quad-tree is a 2^d-ary tree, the kd-tree is a binary tree. This reduces the branching factor at each node and thus the overall storage requirements. At each level down the tree, the point kd-tree divides the set of points in two, alternating the dimensions along which space is split.

The theoretical computational complexity of the point quad-tree and point kd-trees are the same as their region based counter-parts: $O(\log N)$ time for insertion,

[4]It has been suggested that the region quad-trees and the region kd-trees be called quad-tries and kd-tries, to differentiate them from the original point based tree structures they were derived from. However this naming convention never became popular, and at least in computer graphics the unqualified term "quad-tree" usually refers to a (region) quad-trie, and not the original (point) quad-tree. To distinguish between the two data structures (quad-trees and quad-tries), We use the term "quad-tree" preceded by the qualifiers "region" (a trie) and "point" (the original tree).

and point existence queries, $O(N \log N)$ time to build the tree, and $O(N)$ memory. A range search query takes $O(N^{1-1/d})$ in the worst case and $O(F + \log N)$ in the average case. However deletion is more complex because the data points also serve to partition the space from which they are drawn.

Range Trees

A third type of tree, the *range tree*, yields better worst-case range search times at the expense of storage and a more complicated implementation (Overmars, 1983; de Berg et al., 1997). A range tree query performs a 1-dimensional search along the first dimension to select a subset of the data. The subset of data is then searched using a 1-dimensional search along the second dimension to find a smaller subset of the data. This procedure is recursively applied until all dimensions are exhausted. Range trees require $O(N \log^{d-1} N)$ storage, $O(N \log^{d-1} N)$ operations to build, and provide a worst case range search time of $O(\log^d N + F)$. Layered range trees (de Berg et al., 1997) use fractional cascading to reduce the search query time by a factor of $O(\log N)$ to $O(\log^{d-1} N + F)$, while maintaining the same storage requirements. The worst case range search time is better than the quad-tree or kd-tree worst case, but in practice the quad-tree and kd-tree will perform as well, or better (Overmars, 1983).

Summary

Region based and point based tree structures provide an elegant approach to solving the range search problem. Point based trees tend to be simpler to code and maintain, because:

- The partitioning of space is implicit in the data points rather than explicitly specified as required by a region based tree.
- Leaf nodes and interior nodes can be the same data type.
- One does not have to maintain and search buckets of data.

However, when searching the tree for data points, a region based tree may require slightly less computation. This is because the use of buckets reduces the depth of the tree and thus the time required to find the relevant data points. One should note, however, that for certain data sets the region based trees may result in highly unbalanced trees. Kd-trees have the advantage that the algorithm is independent of the dimensionality of data, allowing one to write a single kd-tree implementation for both 2D and 3D simulations. Also a well designed implementation can be used for searching arbitrary types of k-dimensional data, not just point based data. It has also been argued that the kd-tree branching factor of two is preferable to the 2^d branching factor of a quad-tree, because this reduces the memory cost of interior nodes (Samet, 1989). From a theoretical point of view, the region based trees, the point based trees, quad-trees, and kd-trees exhibit similar computation and memory costs. Range trees provide better worst cast performance at the expense of memory and a more complicated implementation, but in practice the simpler tree based data structures will perform as well or better.

7.2.6 Discussion

A fundamental problem encountered in the implementation of dynamically coupled particle system is the problem of finding the neighbors for each particle. This problem appears in the computation of forces, in the particle creation heuristics, in surface triangulation, and in the use of modeling tools. The problem is formalized in Section 7.2.1 as the range search problem. To solve the range search problem we have compared the use of hashing, fixed grids, reduced grids, and hierarchical based data structures.

Hashing, a technique for efficiently storing and retrieving data, is ineffective for computing such spatial relationships. The fixed grid data structure, which partition space in equal size cells, is a good choice for volume modeling, though inefficient in memory for surface modeling. The reduced grid data structure overcomes the memory problem of fixed grids, though search performance may be poor for data sets aligned with the major axes and for data sets with spatially periodic structure. Since our particles will be uniformly distributed with respect to the object being modeled, hierarchical structures are good for both volume and surface based particle systems. The uniform distribution will tend to create well balanced trees resulting in efficient access. When the cell bucket size is chosen wisely, the region-based trees maybe be faster than point-based trees, as the time needed to traverse farther down a point-based tree will outweigh the extra distance tests that will be included in a region-based tree search query. If we use uniform splitting of regions, a region-based tree structure can produce the same partitioning of space as a fixed grid, but with varying resolution. Thus, we can use arguments similar to those presented for the fixed grid to select a leaf cell bucket size.

The best data structure to use will ultimately depend on the given application, and whether one wants to favor generality or speed. In practice, we use a region kd-tree with uniform splitting along each dimension. To further reduce computation, we perform this operation occasionally and cache the list of neighbors for intermediate time steps. We use the kd-tree for both our surface and volume modeling research. Our implementation is independent of the dimension of the data, the size of the data set, the data types, and the distribution of data. While we chose a kd-tree for generality, a production system may wish to trade generality for a strategy finely tuned to the problem at hand. For such a finely tuned systems, we would suggest a region tree or reduced grid for surface modeling, and a region tree, fixed grid, or reduced grid for volume modeling.

7.3 Numerical Integration

In this section we discuss methods for integrating the forces of our particle system over time to arrive at a new system state consisting of a position and orientation for each particle.

7.3.1 Equations of Motion

Having resolved our method of efficiently computing the forces, torques, and the neighbors for a system of particles, we now turn to the problem of integrating the forces through time to compute the time-varying positions and orientations. To create materials which behave as real-world materials, with momentum and the transfer of momentum between objects, we integrate a second order dynamics system. This is appropriate for applications such as physics-based animation where the motion of the objects over time, is more interesting than the final energy minimum. On the other hand, a first order dynamics system is more appropriate for many applications which solve an optimization criterion, such as surface fitting to data samples, For this case, it is adequate to assign the computed forces to the velocity vectors and solve for the change in positions. This is a straight forward simplification and not discussed further. We will discuss the more complex problem of integrating a second order dynamical system.

An un-oriented particle system is governed by (3.1). An oriented particle system is governed by (3.1) and (4.1). Including the model of heat, the inter-particle force terms are functions of the form

$$\mathbf{f}_i^{int} = \mathbf{f}_i^{int}(\mathbf{x}_1, \mathbf{q}_1, \psi_1, \mathbf{x}_2, \mathbf{q}_2, \psi_2, \ldots, \mathbf{x}_N, \mathbf{q}_N, \psi_N)$$

and the inter-particle torque terms are of the form

$$\boldsymbol{\tau}_i^{int} = \boldsymbol{\tau}_i^{int}(\mathbf{x}_1, \mathbf{q}_1, \mathbf{x}_2, \mathbf{q}_2, \ldots, \mathbf{x}_N, \mathbf{q}_N),$$

which is identical to equation (4.4). The external force and torque terms $\mathbf{f}_i^{ext}$ and $\boldsymbol{\tau}_i^{ext}$ are functions of individual particle positions, particle orientations, and external state variables and are given by the Lagrangian equations of motion (4.3), and (4.5).

An optimal solution strategy depends on the type of equations we are trying to integrate. Equations (3.1) and (4.1) are second-order non-linear ordinary differential equations. The force and torque functions introduce highly non-linear terms making this a difficult problem to solve. In addition, the torques are dependent on both particle orientations and positions, thus coupling (3.1) and (4.1) together. We characterize our problem as an initial value problem on two sets of coupled second order non-linear differential equations, with the goal of determining the values of the dependent variables $\mathbf{x}_i$ and $\mathbf{q}_i$, for $i = 1, 2, \ldots, N$, at a set of values in the independent variable t.

The standard approach is to first reduce the second order system of differential equations into two coupled sets of first order differential equations, and then numerically approximate the unknown dependent variables using a finite difference based scheme (Press et al., 1992). For example (3.1) can be rewritten as

$$\frac{\mathrm{d}\mathbf{x}_i}{\mathrm{d}t} = \mathbf{v}_i \qquad (7.2)$$

$$\frac{\mathrm{d}\mathbf{v}_i}{\mathrm{d}t} = \frac{\mathbf{f}_i}{m_i} \tag{7.3}$$

$$\mathbf{f}_i = \mathbf{f}_i^{int} - \mathbf{f}_i^{ext} - \gamma_i \mathbf{v}_i, \tag{7.4}$$

where the velocity $\mathbf{v}_i$ is a new state variable. An equivalent reduction exists for (4.1) Using numerical techniques, the values of $\mathbf{x}_i$ and $\mathbf{v}_i$ can then be advanced from their values at time t to their new values at time $t + \Delta t$ for some value Δt.

7.3.2 Overview of Integration Methods

To solve the initial value problem, all numerical techniques discretize the changes in the dependent and independent variables as finite steps, e.g. $\Delta t, \Delta\mathbf{x}_i$, and $\Delta\mathbf{q}_i$. Doing so allows one to represent the differential changes as algebraic equations composed of these finite steps. As the step size is made small, a good approximation to the underlying differential equation is achieved. There are two approaches to constructing such algebraic equations. Explicit schemes produce new dependent variable values explicitly from the algebraic equations in terms of previous dependent variable values. Implicit schemes produce the new values implicitly in terms of both previous and new dependent variable values.

Implicit schemes applied to linear systems are guaranteed to be stable for any step size, and for non-linear systems are known to generally exhibit good stability. The computational complexity required at each time step is their major drawback. Implicit formulations are easily solved by matrix factorization for linear systems of equations. However, non-linear systems of equations have to be solved iteratively at each step, which can be extremely slow compared to explicit methods. Semi-implicit methods result when one linearizes a non-linear system of equations, and then iteratively solves the linearized version. This still requires matrix factorization, perhaps more than once per time step.

Writing the particle system dependencies as a matrix will result in an $M \times M$ matrix, where M is the number of particles times the number of degrees of freedom per particle. Assuming a particle system with nearby neighbor dependencies, the corresponding matrix will be unstructured and sparse. In fact, the positions of the non-zero elements will be continually changing due to the dynamic coupling inherent in our system. While there exist specialized techniques for inverting certain subclasses of sparse matrices, such as banded diagonal matrices, unstructured sparse matrices will require more general techniques. One could use standard techniques such as Gauss-Jordan elimination or LU decomposition, both requiring $O(N^3)$ time[5]. Since our sparsity pattern is irregular one may want to use matrix techniques designed for sparse linear systems (Press et al., 1992), to reduce the fill-in that occurs during matrix inversion. Such techniques have three steps: (1) An analyze step is done once for each non-zero pattern, (2) a factorize step is computed for each matrix fitting the pattern, and (3) an operate step is computed for each new right-hand side of a matrix equation. Since our particle couplings will be dynamically changing, the

[5]Matrix inversion software solutions in general require $O(N^3)$ time, however it has been shown that inversion can be computed in $O(N^{\log_2 7})$ (Press et al., 1992, page 104).

analyze portion, the most computationally expensive of the three steps, will need to be repeatedly computed along with the other two steps. Relaxation methods can also be used to speed up the convergence of a solution to a matrix equation. Multigrid relaxation methods (Terzopoulos, 1986) alone and combined with conjugate gradient descent algorithms (Szeliski, 1990) have been shown to decrease the computational time needed to find minimal energy states for optimal surface fitting problems in computer vision.

Explicit schemes are popular because new values can be computed directly from previous values. However, unless care is taken, they have a tendency to become unstable or require a small step size. There are a variety of explicit schemes to choose from depending on accuracy and efficiency. According to Hockney and Eastwood (1988),

> The compromise between accuracy and efficiency can be altered in two ways - either by using a higher-order scheme and larger time step or by using a lower-order scheme and smaller time step. The former approach suffers because (1) the time step is limited by natural frequencies of the system, (2) higher-order schemes often have more restrictive stability limits on the time step, and (3) high-order schemes need force values at several time levels. Usually, the best compromise between accuracy, stability, and efficiency in *many-body calculations* is found by using a simple second-order accurate schemes (such as leapfrog) and adjusting the time step accordingly.

7.3.3 Integration

We have chosen to integrate the system of particles using low order explicit schemes primarily because our focus has been on applying our system to a variety of problems including *interactive* surface modeling. While explicit schemes have drawbacks, implicit or semi-implicit scheme will generally be slower than linear time, making these unsuitable for interactive applications. We have used both the Euler and the leapfrog integration methods. Both approaches require a single evaluation of the force equations for a given time step.

Explicit Euler

At each time step we sum the all of the forces acting on each particle and integrate over the time interval. We solve our second order system of differential equations as two coupled first order systems. We have used Euler's method for its simplicity (Press et al., 1988). However instead of using Euler's method directly on both systems we can make a better estimate for the second system by using the average of the old and new velocities computed by the Euler step on the first system (Gould and Tobochnik, 1988). Both the Euler and the modified Euler method have local truncation error of $O(h^2)$ at each step and are first order accurate integration schemes.

We designate time values by superscripts, defining t as the current time and integrate over the time interval $h = \Delta t$ to time $t + h$. We use the equations of motion from Appendix B that are simplified for our choice of inertia tensor. The translational equations are

$$\begin{aligned}\mathbf{a}^t &= \frac{\mathbf{f}^t}{m}\\ \mathbf{v}^{t+h} &= \mathbf{v}^t + h\,\mathbf{a}^t\\ \mathbf{x}^{t+h} &= \mathbf{x}^t + h\frac{\mathbf{v}^{t+h}+\mathbf{v}^t}{2}.\end{aligned}$$

and angular motion equations are

$$\begin{aligned}\mathbf{b}^t &= \mathbf{I}^{-1}\boldsymbol{\tau}^t\\ \boldsymbol{\omega}^{t+h} &= \boldsymbol{\omega}^t + h\,\mathbf{b}^t\\ \boldsymbol{\theta} &= h\frac{\boldsymbol{\omega}^{t+h}+\boldsymbol{\omega}^t}{2}\\ \mathbf{q}_{\boldsymbol{\theta}} &:= \left[\cos\left(\|\boldsymbol{\theta}\|/2\right), \hat{\boldsymbol{\theta}}\sin\left(\|\boldsymbol{\theta}\|/2\right)\right]\\ \mathbf{q}^{t+h} &= \mathbf{q}^t\mathbf{q}_{\boldsymbol{\theta}}.\end{aligned}$$

For angular motion, the change in orientation $\boldsymbol{\theta}$ is computed from the angular velocity $\boldsymbol{\omega}$, where both are represented as vectors. The incremental rotation $\boldsymbol{\theta}$ is then converted to the quaternion $\mathbf{q}_{\boldsymbol{\theta}}$ (Shoemake, 1989), before updating the particle orientation through quaternion multiplication[6]. To avoid a loss of accuracy, due to finite floating point representation, we normalize the quaternion after each step.

Leapfrog

The leapfrog scheme is defined by the time centered finite differences

$$\frac{x^{t+h}-x^t}{h} = v^{t+h/2}$$

and

$$\frac{v^{t+h/2}-v^{t-h/2}}{h} = a^t,$$

defined here for a one-dimensional space. It is similar to the Euler method in that the same amount of work is required and no auxiliary storage is needed, as is required in higher order methods. The leapfrog scheme differs from the explicit Euler in that the velocity values are defined at the midpoint between time steps. The result is a local truncation error at each step of $O(h^3)$ making this a second order accurate integration scheme. A differential equation is time-reversible if a particle integrated forwards in time in a given force field will retrace its path and return to the starting point, when integrated backwards in time. Time-reversible difference approximations are obtained by defining time-centered derivatives, such as the leapfrog scheme (Hockney and Eastwood, 1988).

[6]Note, the vector $\hat{\boldsymbol{\theta}}$, in the vector to quaternion conversion, is the unit vector in the direction $\boldsymbol{\theta}$.

Using the same notation as for the Euler equations, the leapfrog equations for translational motion are

$$\begin{aligned} \mathbf{a}^t &= \frac{\mathbf{f}^t}{m} \\ \mathbf{v}^{t+h/2} &= \mathbf{v}^{t-h/2} + h\,\mathbf{a}^t \\ \mathbf{x}^{t+h} &= \mathbf{x}^t + h\,\mathbf{v}^{t+h/2}. \end{aligned}$$

The equations for angular motion are

$$\begin{aligned} \mathbf{b}^t &= \mathbf{I}^{-1}\boldsymbol{\tau}^t \\ \boldsymbol{\omega}^{t+h/2} &= \boldsymbol{\omega}^{t-h/2} + h\,\mathbf{b}^t \\ \boldsymbol{\theta} &= h\,\boldsymbol{\omega}^{t+h/2} \\ \mathbf{q}_{\boldsymbol{\theta}} &:= \left[\cos\left(\|\boldsymbol{\theta}\|/2\right), \hat{\boldsymbol{\theta}}\sin\left(\|\boldsymbol{\theta}\|/2\right)\right] \\ \mathbf{q}^{t+h} &= \mathbf{q}^t\mathbf{q}_{\boldsymbol{\theta}}. \end{aligned}$$

Unfortunately leapfrog does not compute v^{t+h}, but rather $v^{t+h/2}$. If v^{t+h} is needed, such as to compute a damping force, an approximation can easily be computed as follows. We construct the two formula

$$\mathbf{v}^t = \frac{\mathbf{v}^{t-h/2} + \mathbf{v}^{t+h/2}}{2}$$

and

$$\mathbf{v}^{t+h/2} = \frac{\mathbf{v}^t + \mathbf{v}^{t+h}}{2}.$$

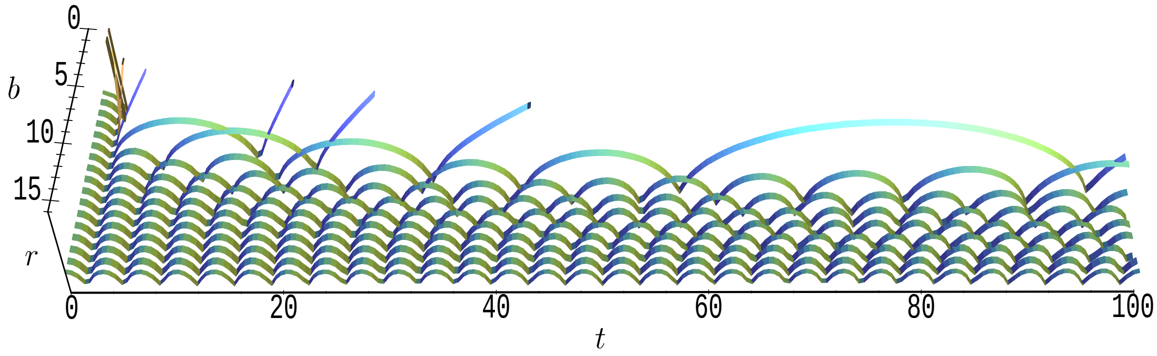
to describe the velocity at times t and $t + h/2$ as average velocities. Solving for $\mathbf{v}^{t+h}$ results in the extrapolation

$$\mathbf{v}^{t+h} = \frac{3\mathbf{v}^{t+h/2} - \mathbf{v}^{t-h/2}}{2}. \tag{7.5}$$

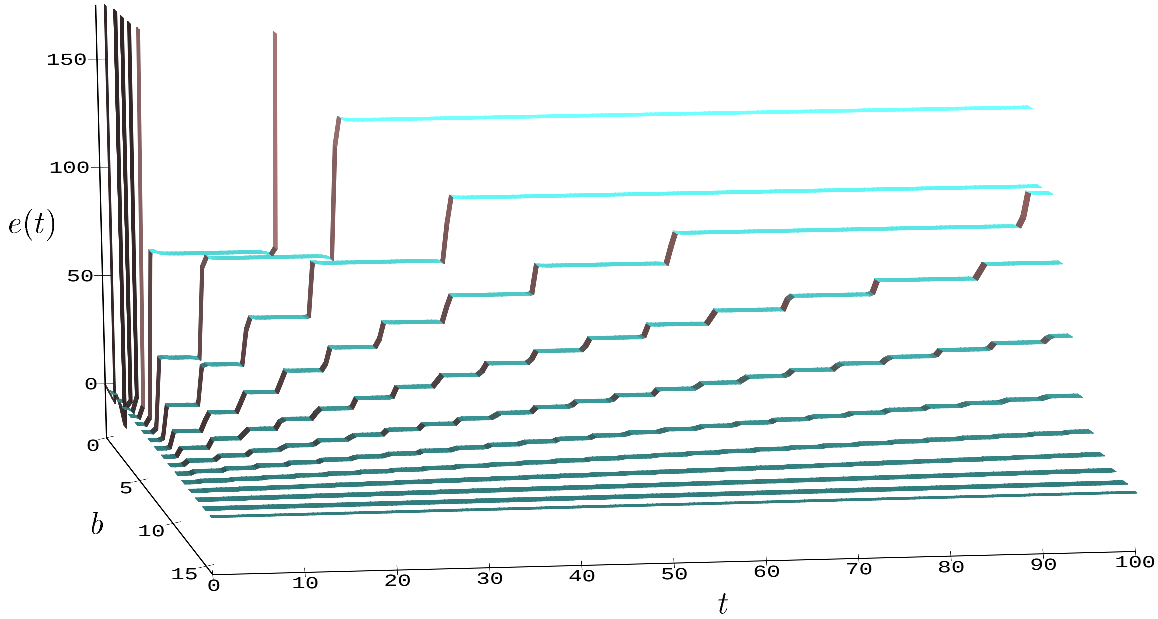
The same method can also be used to extrapolate the angular velocities.

Comparison

To provide a comparison between the various explicit integration schemes we prepared two tests which we executed for all three integration schemes over a variety of time steps. The first test is a simple two particle system without damping which allowed us to compare the stability and accuracy of integration. Without damping the particle system should conserve energy and thus we could measure the accuracy based on the relative error in the system energy. The second test used a complex system of 800 particles with both global velocity damping and relative inter-particle velocity damping. Since the energy in the system should decrease due to the damping terms, this test does not allow us to measure the accuracy using the relative error in system energy. However it does provide complex n-body interactions as we might expect in modeling and animation.



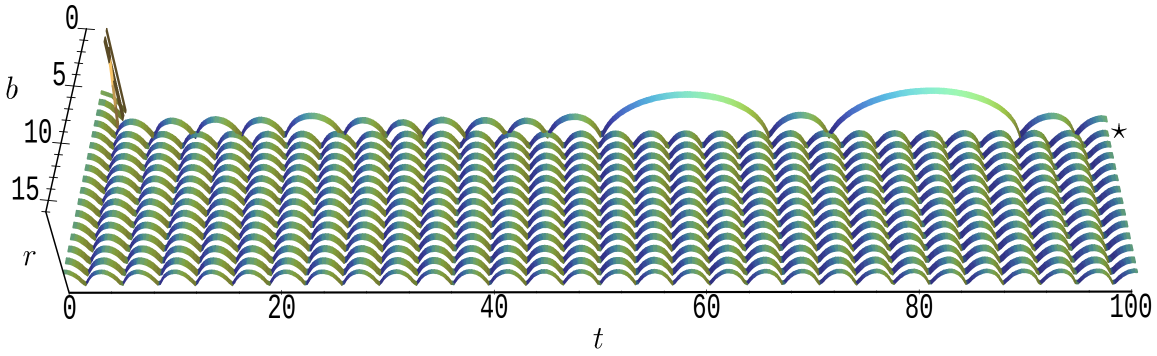
(a) Particle separation for a two particle system



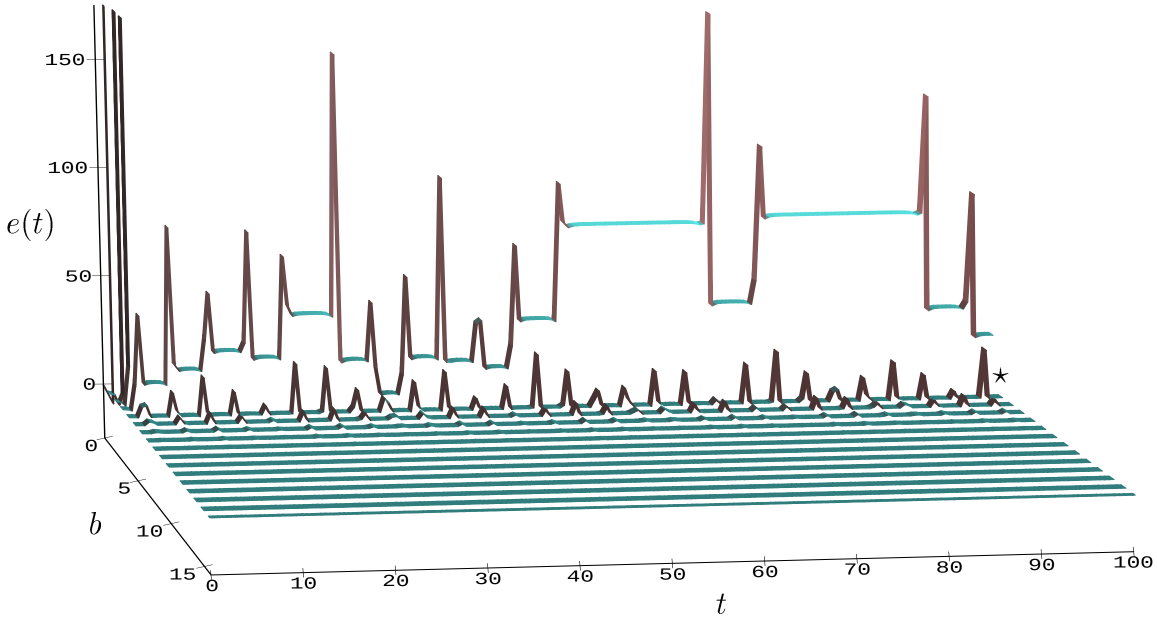
(b) Relative error of total system energy expressed as a percentage

Figure 7.2: Empirical results of modified Euler integration

The axes of the plots are as follows. Left to right is time t and ranges from 0 to 100. Back to front is logarithmic in the time step $h = 2^{-b}$ and ranges from 2^0 to 2^{-16}. (a) The vertical axis is particle separation $r = \|\mathbf{x}_1 - \mathbf{x}_2\|$ and ranges from 0 to 8. (b) The vertical axis is relative error e in system energy and ranges from -25% to 175%. For plotting, the data was reduced to 1601 samples for each simulation.



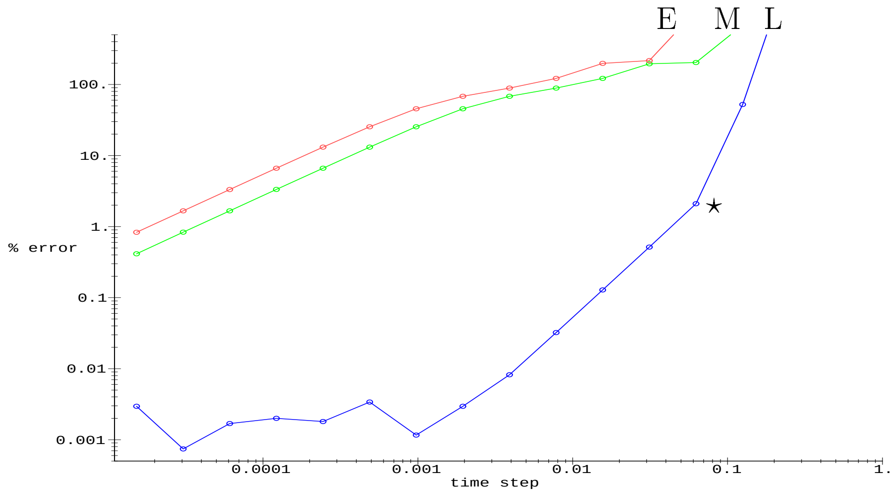
(a) Particle separation for a two particle system



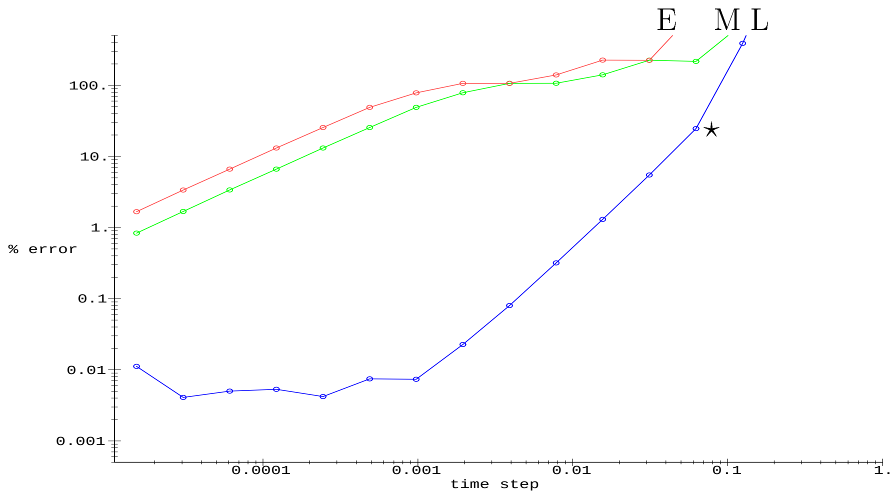
(b) Relative error of total system energy expressed as a percentage

Figure 7.3: Empirical results of Leapfrog integration

The axes of the plots are as follows. Left to right is time t and ranges from 0 to 100. Back to front is logarithmic in the time step $h = 2^{-b}$ and ranges from 2^0 to 2^{-16}. (a) The vertical axis is particle separation $r = \|\mathbf{x}_1 - \mathbf{x}_2\|$ and ranges from 0 to 8. (b) The vertical axis is relative error e in system energy and ranges from -25% to 175%. For plotting, the data was reduced to 1601 samples for each simulation.



(a) Average relative error in system energy



(b) Maximum relative error in system energy

Figure 7.4: Relative error in system energy

The horizontal axis is logarithmic in the time step h. (a) The vertical axis is logarithmic in the average relative system error over the simulation. (b) The vertical axis is logarithmic in the maximum relative system error over the simulation. The Euler method is displayed in red and labeled "E". The modified Euler method is displayed in green and labeled "M". The Leapfrog method is displayed in blue and labeled "L". The unusual results for the leapfrog for small relative error values ($e \approx 0.01\%$) we believe are due to limited machine precision in calculating the error measure. The $\star$ data points corresponds to the simulations with $\star$ symbols in Figure 7.3.

Comparison: First Test

For the first test the particle system studied was as follows:

- Particles interact according to the unweighted Lennard-Jones potential.
- Lennard-Jones parameters of $r_o = 1$, $n = 4$, $m = 2$, and $e = 1.0$.
- Two particles initially separated by 2 units.
- No neighborhood bounds, that is the particles are always coupled.
- No damping.

The system was integrated under the following conditions.

- For the Euler, modified Euler, Leapfrog explicit integration schemes.
- For times steps $h = 2^{-b}$ where $b = 0, 1, 2, \ldots, 16$.
- From time $t = 0$ to $t = 100$.

For each test case and for each time step executed we collected data on the particle separation, kinetic energy, and potential energy. The results from these simulations are displayed in Figures 7.2, 7.3, and 7.4.

In Figures 7.2(a) and 7.3(a) show the particle separation plotted as a height field for the modified Euler and Leapfrog schemes respectively. Time runs linearly, left to right. The time step is decreasing, back to front, on a logarithmic scale. The Euler method is not shown as it is not substantially different than the modified Euler, except for a factor of two as is explained shortly. Note that the modified Euler method is still converging to a solution for the smallest time step. This can be noticed by visually inspecting the right hand side of Figure 7.2 where differences in the period of oscillation can be seen.

Figures 7.2(b), and 7.3(b) display the relative error in the total system energy. The relative error e is computed as the percentage difference in system energy at time t to original system energy

$$e(t) = \frac{E_S(t) - E_S(0)}{E_S(0)} \times 100$$

where $E_S(0)$ is the sum of kinetic and potential energies at time 0

$$\begin{aligned} E_K(0) &= 0 \\ E_P(0) &= \phi_{\mathrm{LJ}}(2) = -0.4375 \end{aligned}$$

and $E_S(t)$ is the sum of kinetic and potential energies at time t

$$\begin{aligned} E_K(t) &= \frac{1}{2}\left(m_1\|\mathbf{v}_1(t)\|^2 + m_2\|\mathbf{v}_2(t)\|^2\right) \\ E_P(t) &= \phi_{\mathrm{LJ}}(\|\mathbf{x}_1(t) - \mathbf{x}_2(t)\|). \end{aligned}$$

Figure 7.4(a) displays the average value of the relative error over the entire run of each simulation. Figure 7.4(b) displays the maximum value of the relative error over the entire run of each simulation. For both cases the absolute value of the relative error is used.

The differences between the Euler method and the modified Euler is as expected. Analysis using a Taylor series expansion shows that the second step of the modified Euler method has half the local truncation error than does the second step of the Euler method. This is noticeable in the results. The data shows the modified Euler with a time step of $2h$ produces results almost identical to the Euler method with time step of h. Both Euler methods have a local truncation error of $O(h^2)$ per step.

The Leapfrog method performed well in practiced. As the time step decreased, it quickly converged to a stable solution. Analysis using a Taylor series expansion shows Leapfrog exhibits a local truncation error of $O(h^3)$ per step and thus global truncation error of $O(h^2)$. The rate of decrease of error in the data, matches this analysis.

When viewing the raw data for the various integration schemes, at stable time steps, the Euler methods almost exclusively exhibited positive relative error, though over small intervals it exhibited negative relative error. The Leapfrog exhibited both positive and negative relative errors in nearly equal amounts. We speculate the difference is due to the nature of the potential energy function combined with the fact that the Leapfrog is a time centered scheme, while the Euler methods are not. However, the exact reason for these difference requires further study.

Comparison: Second test

For the second test the particle system studied was as follows:

- A particle system containing 800 particles.
- Particles interacting according to the weighted Lennard-Jones potential, global velocity damping, viscous inter-particle damping, and limited particle interaction.
- The parameters of the Lennard-Jones potential were $r_o = 1$, $n = 4$, $m = 2$, and $e = 1$.
- The parameters of the weighting function were $r_a = r_o$ and $r_b = 1.7r_o$.
- Neighborhood range was set to $1.7r_o$.
- Neighbors were recomputed after every time step.
- The global velocity based damping was $-0.25\dot{\mathbf{x}}_i$.
- The inter-particle viscous damping was $-0.125\,(\dot{\mathbf{x}}_i - \dot{\mathbf{x}}_j)$.
- In the case of the Leapfrog integration scheme, extrapolated velocities defined by (7.5) were used in the damping computations.

We ran the test on a particle system of 800 particles that was initially positioned on a hexagonal grid of $8 \times 10 \times 10$ with particles separated by $1.5r_o$. The separation value was picked to create a large initial system energy. To provide for more complex behavior, the positions were slightly displaced by random amounts. This removed the strong spatial symmetry of the initial placement. The displacement was computed as a vector of three independent observations of a Gaussian random variable with zero mean and a variance of 0.05. If the magnitude of the displacement was greater than the variance, then the displacement vector was scaled to be of length equal to the variance. This was to keep particles receiving a large random displacement from deviating too far from their initial position.

The system was integrated under the following conditions:

- For the Euler, modified Euler, Leapfrog explicit integration schemes.
- For times steps $h = 2^{-b}$ where $b = 3, 4, 5, 6, 7$. This is equivalent to the range $h = 0.125$ to $h = 0.0078125$.
- From time $t = 0$ to $t = 20$.

The initial configuration for the tests is shown in Figure 7.5. The particles are color coded based on the number of interacting neighbors. Particles on the edge boundaries have fewer neighbors and are darker in color. Some particles on the face have more neighbors than others due to the initial random displacements. The lines drawn between particles are the neighbor connections.

For each test case and for each time step executed, we collected data on the system kinetic energy and system potential energy. The results from these simulations are displayed in Figures 7.6, 7.7, 7.8, and 7.9. The system energies are plotted in the left column and are labeled by (a), (c), and (e). Images from the final frame of each animation are displayed next to the energy plots and are labeled by (b), (d), and (f). The results for each time step are shown on the same page. The results are presented in the order of increasing time step size.

The first set of results is for the time step of $\Delta t = 2^{-7}$ (shown in Figure 7.6) and show all three integration schemes as being stable. The initial cube of particles collapses into a ball as the system energy is minimized. The plots of energy show steadily decreasing potential energy (the lower line of each plot) indicating this is a highly damped system. The difference between the total system energy (top line of each plot) and the potential energy is the amount of kinetic energy in the system. The negative total potential energy (and hence system energy) is due to the definition of the Lennard-Jones potential. For the same time step in the two-particle undamped system, the Euler method was unstable. The inclusion of damping has stabilized the Euler method in this test, even though the interactions are more complex.

The second set of results (shown in Figure 7.7) is for the same initial configuration, yet the a time step was twice as large. Both the Leapfrog and modified Euler method produce similar stable results, but the Euler method exhibits instabilities. These are seen as large spikes in the system energy plot. The large potential energy values indicate that particles came closer than the collision distance. Finally, the flattening

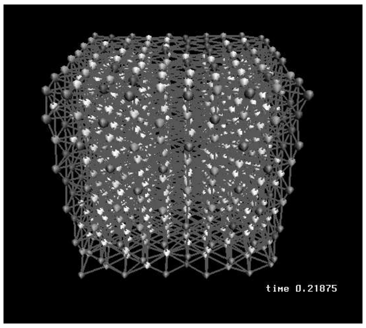


Figure 7.5: Initial particle system configuration

This is the initial particle system configuration of 800 particles arranged on an hexagonal grid with random displacements assigned to each particle.

---

of the potential energy curve to near zero energy indicates that only a few of the particles are interacting with nearby particles.

The third set of results (shown in Figure 7.8) is for time step twice as large again. Both Euler and modified Euler methods exhibit instabilities, with the Euler method becoming unstable almost immediately. The modified Euler method became unstable at about the same time that the Euler method became unstable in the previous experiment. The Leapfrog method remained stable.

The fourth set of results (shown in Figure 7.9) is for a time step twice as large again. In this case all three integration methods became unstable. In the two particle system, the Leapfrog scheme was stable at this time step. Since the interactions in this system are much more complex, this is not a surprising result.

In summary, although the Euler and modified Euler method performed better with the inclusion of damping, the Leapfrog stability limit was still four times that of the Euler method, and twice that of the modified Euler method.

Additional measures

To allow us to take larger time steps, then would be otherwise be allowable, we can apply the following measures.

- We can introduce limits on the maximum velocity and maximum change in velocity.
- We can place limits on the inter-particle forces, such as clipping the magnitude of Lennard-Jones force for distances less than the collision distance to the value at the collision distance.

These measures in effect reduce the natural frequencies of the system resulting in stable integration at larger time steps.

From our testing it became obvious that the majority of error for the Euler method was introduced when integrating over regions with large changes in force. Since the

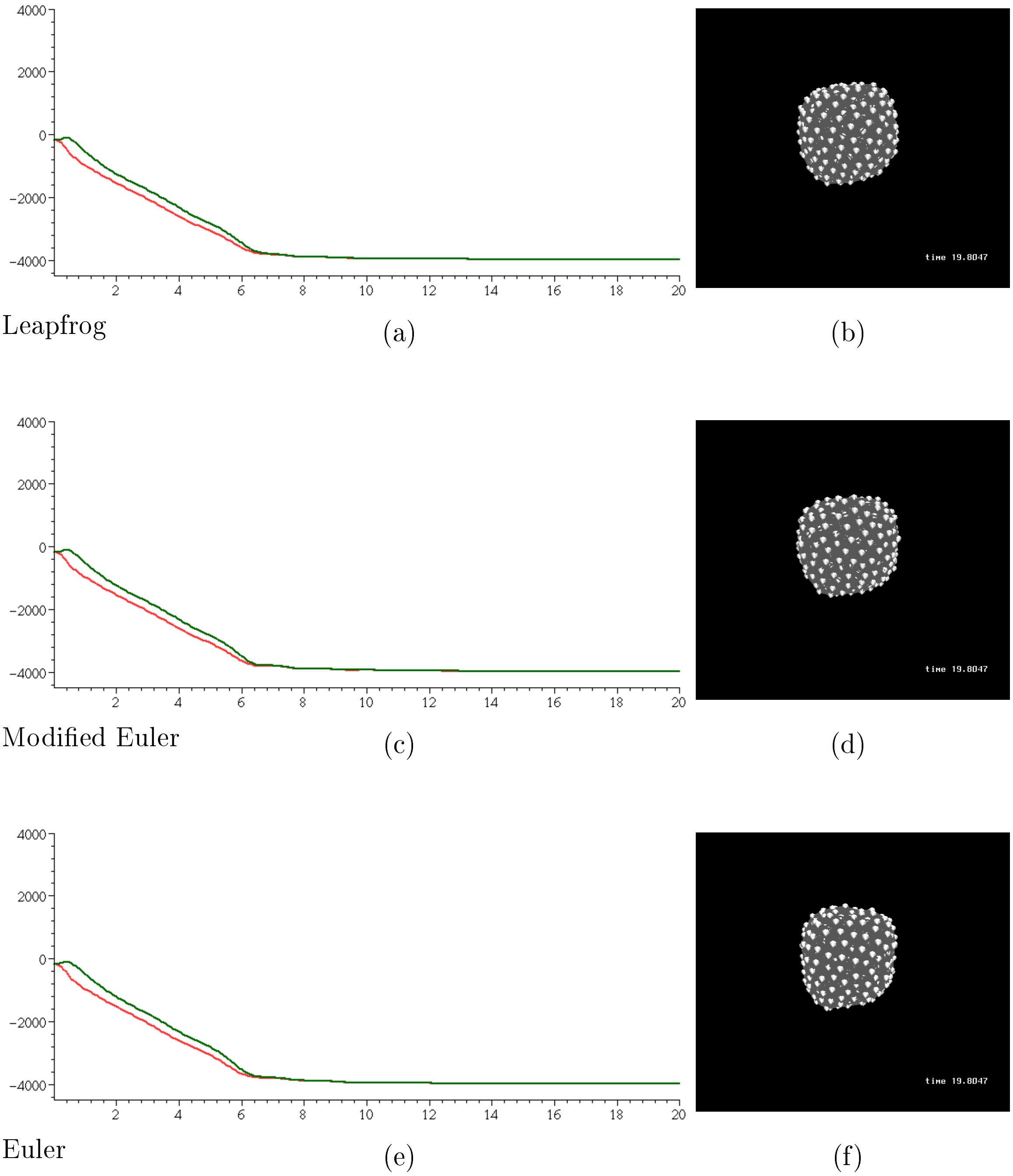
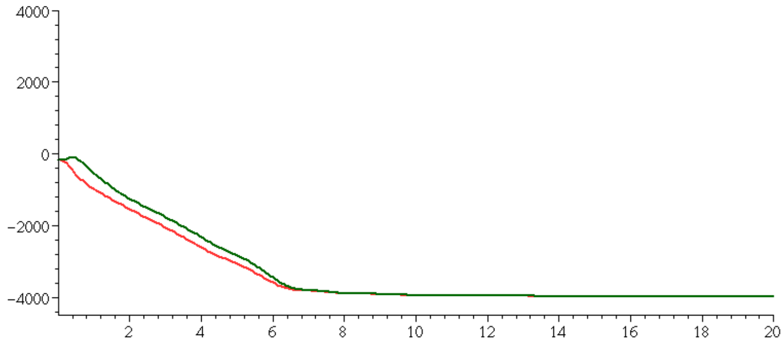
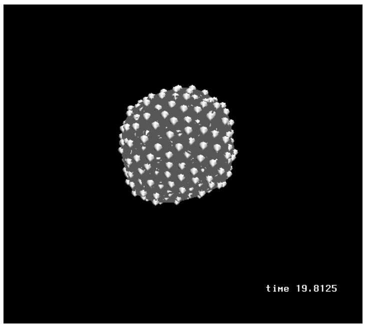


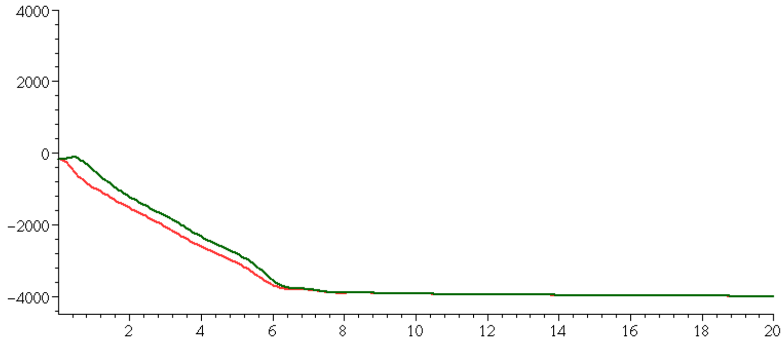
Figure 7.6: Integration with $\Delta t = 2^{-7}$.

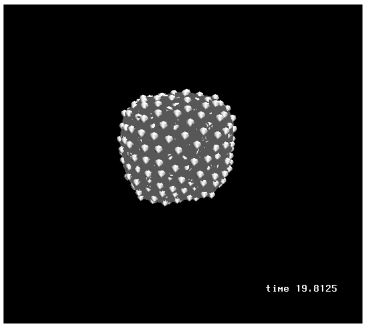
The rows above display the results for Leapfrog, Modified Euler, and Euler schemes respectively. Left column lists the plots of system energy versus time. The right column displays the final image from each animation. The dark line (top/green) is the total system energy $E_S = E_P + E_K$ and the lighter line (bottom/red) is the system potential energy E_P. Note $E_S(t) \geq E_P(t)$. For this time step, all the integration schemes were stable.



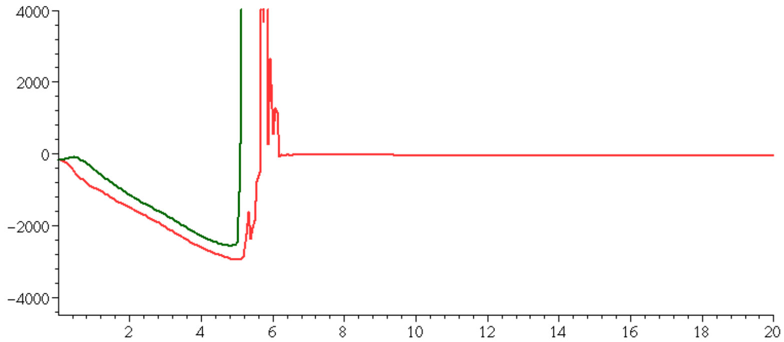


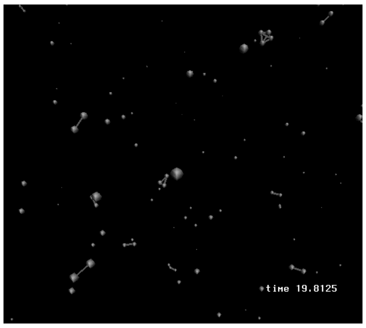
Leapfrog (a) (b)





Modified Euler (c) (d)

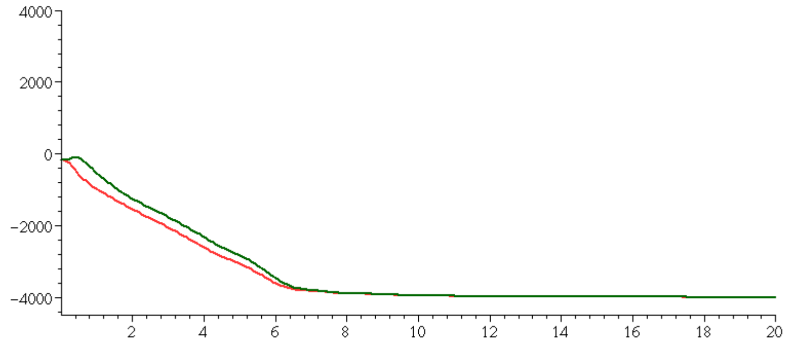


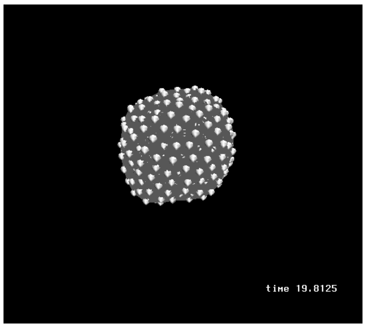


Euler (e) (f)

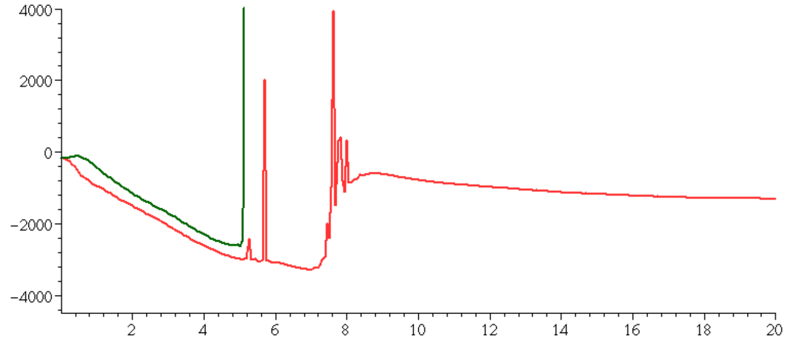
Figure 7.7: Integration with $\Delta t = 2^{-6}$.

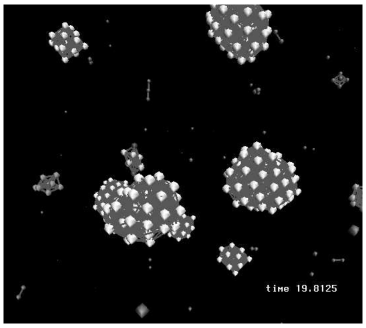
The rows above display the results for Leapfrog, Modified Euler, and Euler schemes respectively. Left column lists the plots of system energy versus time. The right column displays the final image from each animation. The dark line (green) is the total system energy $E_S = E_P + E_K$ and the light line (red) is the system potential energy E_P. Note $E_S(t) \geq E_P(t)$. The spikes in the bottom energy plot indicate instabilities in integration. The result was that the particle system exploded.



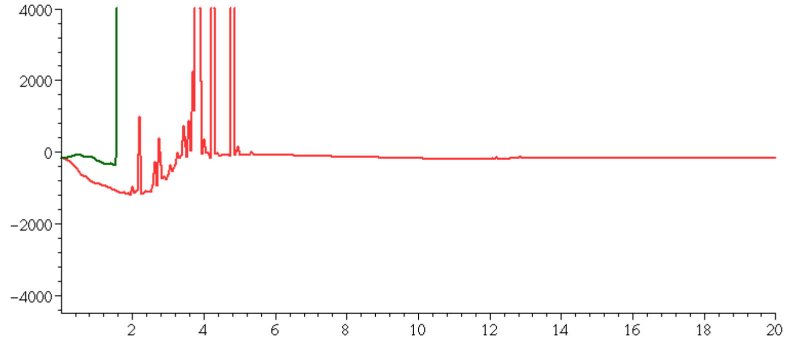


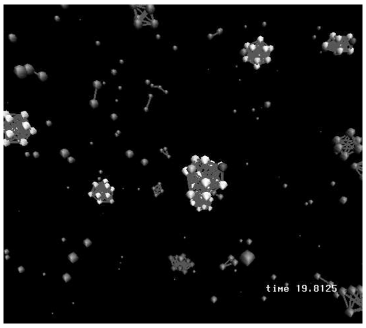
Leapfrog (a) (b)





Modified Euler (c) (d)

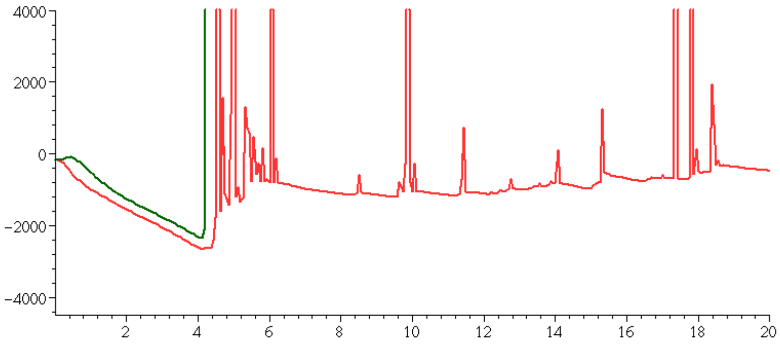


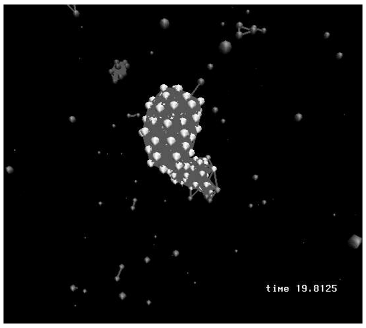


Euler (e) (f)

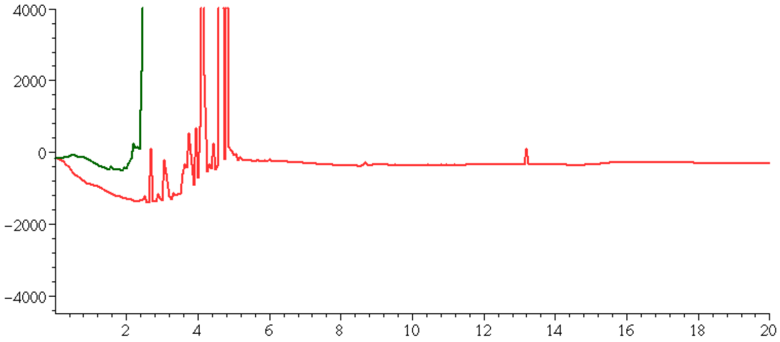
Figure 7.8: Integration with $\Delta t = 2^{-5}$.

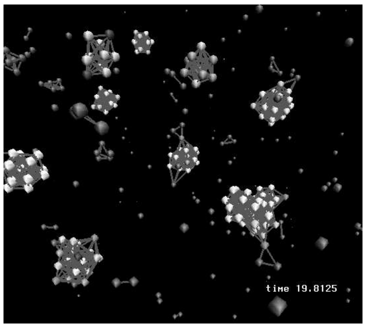
The rows above display the results for Leapfrog, Modified Euler, and Euler schemes respectively. Left column lists the plots of system energy versus time. The right column displays the final image from each animation. The dark line (green) is the total system energy $E_S = E_P + E_K$ and the light line (red) is the system potential energy E_P. Note $E_S(t) \geq E_P(t)$.



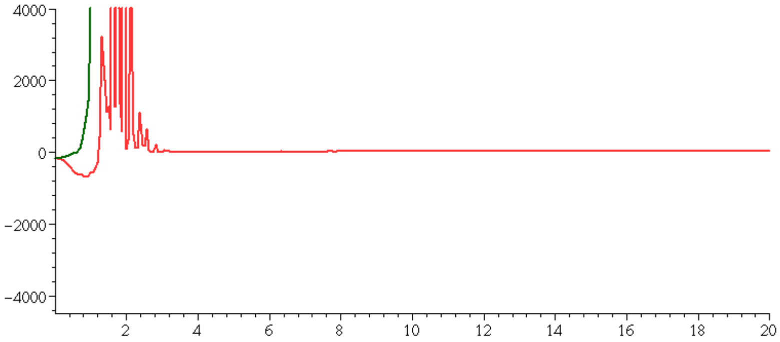


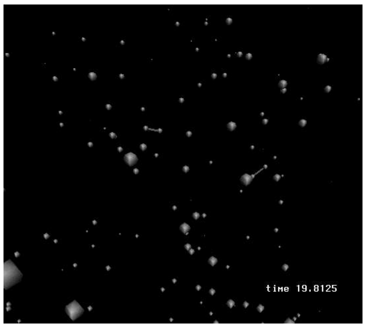
Leapfrog (a) (b)





Modified Euler (c) (d)





Euler (e) (f)

Figure 7.9: Integration with $\Delta t = 2^{-4}$.

The rows above display the results for Leapfrog, Modified Euler, and Euler schemes respectively. Left column lists the plots of system energy versus time. The right column displays the final image from each animation. The dark line (green) is the total system energy $E_S = E_P + E_K$ and the light line (red) is the system potential energy E_P. Note $E_S(t) \geq E_P(t)$.

repulsive force between particles, due to the Lennard-Jones potential, is stronger than the attractive forces, more error tends to be introduced when particles are colliding than when they are moving apart. This can be seen in the graph of the relative error for the Euler method (Figure 7.2(b)) as “steps” of increasing error. It can also be seen as spikes in the error plot (Figure 7.3(b)) of the Leapfrog method. The net result is that the system energy increases and the system eventually explodes as the accumulated error exceeds the binding (potential) energy of the system. For modest time steps, the introduction of damping overcomes the accumulation of error, reversing the process. From experience, both global velocity based damping and viscous damping units are effective (Section 3.3.2). The viscous damping unit is superior for stabilizing numerical errors because it is based on the relative velocity between particles and is independent of rigid body motion.

7.3.4 Stability and Accuracy Analysis

Instead of empirically testing for a suitable time step, we can analytically derive stability and accuracy criteria for the leapfrog integration scheme. For simplicity we analyze a two particle system interacting according to the Lennard-Jones potential. This provides a starting point for deriving the stability criteria of more complex systems and thus choosing an appropriate time step.

Stability

Solving for error propagation, (Hockney and Eastwood, 1988, pp. 97–106) derives the stability boundary

$$\omega \Delta t < 2, \tag{7.6}$$

where ω is the highest oscillation frequency of the system: for the second-order differential equation of motion for undamped motion between a pair of particles, when approximated by the time-centered leapfrog scheme. There are two basic assumptions made in deriving the boundary: (1) the solution of the differential equation is oscillatory in nature and (2) the solution is derived for the worst case. The worst case is defined by the highest frequency ω of the system, which occurs at the maximum change in the magnitude of the force

$$\omega^2 = -\frac{1}{m}\frac{\mathrm{d}f}{\mathrm{d}x} \tag{7.7}$$

for negative forces.

The Frequency of Oscillation

For a general force $f(x)$, equations (7.6) and (7.7) allow us to relate the force to the highest oscillation frequency of the system and from that determine the stability and accuracy for a given time step. We determine the time step for a system of particles absent of any damping forces and interacting according to the Lennard-Jones potential energy function. We choose the Lennard-Jones potential because of all our inter-particle potentials it presents the largest magnitude in change of force.

We follow a one-dimensional argument, similar to that given in (Hockney and Eastwood, 1988, pp 461-465). We begin by estimating the highest frequency which occurs when the maximum gradient of the force is present. For the case of the Lennard-Jones potential, this occurs when the neighboring particles move towards each other to a position of closest approach. The equations of undamped motion for two such particles are

$$m_1 \frac{d^2 x_1}{dt^2} = -f(x_2 - x_1) \tag{7.8}$$

$$m_2 \frac{d^2 x_2}{dt^2} = f(x_2 - x_1), \tag{7.9}$$

where f is the magnitude of the repulsive force, x_1 and x_2 are the positions of the two particles, and m_1 and m_2 are the respective masses. We choose to analyze the repulsive force of the Lennard-Jones function, over the attractive force, because the repulsive force exhibits higher values of magnitude. Multiplying the first equation by m_2 and the second equation by m_1, and subtracting yields

$$m_1 m_2 \left(\frac{d^2 x_2}{dt^2} - \frac{d^2 x_1}{dt^2} \right) = m_1 f - m_2 f. \tag{7.10}$$

Reducing, we get the equation of relative motion

$$m^\star \frac{d^2 r}{dt^2} = f(r), \tag{7.11}$$

where $r = x_2 - x_1$ is the particle separation, and $m^\star = m_1 m_2/(m_1 + m_2)$ is the reduced mass. Since f(r) is non-linear, to obtain an effective frequency we consider a small perturbation substituting $r + r'$ for r, where r' is small

$$m^\star \frac{d^2 (r + r')}{dt^2} = f(r + r'). \tag{7.12}$$

We expand the right side as a Taylor series, keeping only the first two terms of the series.

$$f(r + r') \simeq f(r) + r' \frac{\mathrm{d}f}{\mathrm{d}r} + \ldots$$

The left side expands to

$$m^\star \frac{d^2 (r + r')}{dt^2} = m^\star \left(\frac{d^2 r}{dt^2} + \frac{d^2 r'}{dt^2} \right).$$

Expanding (7.12) and subtracting (7.11) gives

$$m^\star \frac{d^2 r'}{dt^2} = r' \frac{\mathrm{d}f}{\mathrm{d}r}. \tag{7.13}$$

Rewriting results in the linearized equation for the perturbation r'

$$\frac{d^2 r'}{dt^2} = -\omega^2 r', \tag{7.14}$$

where

$$\omega = \left(-\frac{1}{m^\star}\frac{\mathrm{d}f}{\mathrm{d}r}\right)^{\frac{1}{2}} \tag{7.15}$$

according to equation (7.7).

When df/dr is negative, as it is for the short-range Lennard-Jones repulsion, equation (7.14) is the differential equation for simple harmonic oscillation with frequency ω given by (7.15). From (7.7) we can see that the maximum frequency will occur for small masses and steeper laws of repulsion between the particles. For the case of the Lennard-Jones potential, the frequency is given by

$$\omega = \left(-\frac{1}{m^\star}\left(\frac{B^\star}{r^{n+2}} - \frac{A^\star}{r^{m+2}}\right)\right)^{\frac{1}{2}}, \tag{7.16}$$

where $A^\star$ and $B^\star$ are the constants

$$A^\star = \frac{\epsilon nm(m+1){r_o}^m}{n-m} \qquad\qquad B^\star = \frac{\epsilon nm(n+1){r_o}^n}{n-m}. \tag{7.17}$$

This expression for ω indicates that as the minimum separation r goes to zero, the frequency of oscillation goes to infinity. Since stability requires $\omega\Delta t < 2$, all time steps, in principle, are unstable. However let us assume that the particles do not come closer than a given distance. For the purpose of this analysis we will assume this distance to be the collision distance σ of the Lennard-Jones potential. In order to prevent the few particles that might end up closer, from becoming unstable, one can limit the force of repulsion for distances less than the collision distance to its value at the collision distance.

We now compute the oscillation frequency for two cases: a worst case at the collision distance, and an average case at the equilibrium spacing. We let the defining parameters of the Lennard-Jones functions be $m = 2$, $n = 4$, $\epsilon = 1$, and $r_o = 1$. And we let the mass for each particle be 1.0 so $m^\star = 0.5$. This results in a frequency for the worst case of 15.0 and for the average case of 4.0.

Accuracy

We now consider the issue of accuracy. Let's assume particles will have an oscillation frequency ω associated with the average separation of r_o. If the leapfrog scheme is used to integrate the equations of motion, then (7.14) is approximated as

$$\frac{r'(t+\Delta t) - 2r'(t) + r'(t-\Delta t)}{(\Delta t)^2} = -\omega^2 r'(t) + \frac{(\Delta t)^2}{12}\frac{d^4 r'}{dt^4} \tag{7.18}$$

based on the Taylor series. For an oscillation at frequency ω, we apply equation (7.14) to find

$$\frac{d^4 r'}{dt^4} = \omega^4 r'. \tag{7.19}$$

The ratio of truncation error (last term on the right-hand side of (7.18)) to the true force (first term on the right-hand side of (7.18)) is

$$\frac{(\omega \Delta t)^2}{12}. \tag{7.20}$$

Based on the above error measure, we again consider two cases: an average case and a worst case. Recall that from equation (7.6), the product of the frequency and time step should be less than 2 for stability. If we let the product equal 0.25 as a condition for sufficient integration for average separation, by (7.15) we can compute ω, and the relative truncation error from (7.20) reduces to 1/2 of a percent. This appears to be of reasonable accuracy for the average case. In the worst case, we assume that few particles will be at the collision limit. Thus we favor a larger time step over increased accuracy. By choosing a product of 1, we get a relative truncation error of 8 percent. A time step satisfying both accuracy and highest frequency constraints is the minimum time step of the two computations.

Having computed the worst case and average case frequencies of the Lennard-Jones function earlier, we now compute time steps. The conditions on the time step can be summarized as follows

- Accuracy at the average separation:

  $r = r_o = 1, \qquad \omega = 4, \qquad \omega\Delta t = 0.25, \qquad \text{error} = 0.5\%, \qquad \Delta t = 0.0625$

- Stability at the highest frequency:

  $r = \sigma = 1/\sqrt{2}, \qquad \omega = 15, \qquad \omega\Delta t = 1.0, \qquad \text{error} = 8\%, \qquad \Delta t = 0.0667$

for the Lennard-Jones parameters $m = 2$, $n = 4$, and $\epsilon = 1$. Our analysis matches the empirical results of Figures 7.2, 7.3, and 7.4. In Figures 7.3 and 7.4 the time step of 0.0625 is marked with a $\star$ symbol. For this time step, the average error was 2.1% over the entire simulation and the maximum error for any step was 24.7%.

7.3.5 Timing Results

To measure the speed at which one can currently calculate a simulation, we ran several tests. Rather than quoting the speed in frames per second, we found it more useful to quote the results in particle pair computations per second. Given a rate of pairs per second, the frame rate for most scenes can be approximated by estimating the number of time steps taken per rendered frame and the number of neighbors per particle. The number of neighbors per particle is dependent both on the neighborhood range and whether the model is using volume or surface particles. The system we tested consisted of 1000 volume particles interacting with their nearest neighbors. The weighting function parameters were $r_a = r_o$ and $r_b = 1.4r_o$, and the neighborhood range was $1.4r_o$. Neighbors were computed every 10 steps. The particles interacted according to the Lennard-Jones potential, inter-particle viscous damping, and global velocity damping. Approximately 19,000 particle pair interactions were computed per second when executing on an SGI O2. That amounts to 6000 pairs per time step, or 3 steps per second [7]. Using the same number of oriented particles would result in approximately 4 to 6 steps per second, depending on the number of surface potentials used. It should be noted that the code executed in this test was not optimized, but rather written for generality. From prior experience, profiling and optimizing the code should give a speed up factor on the order of 4 to 8 times the unoptimized code speed. Such optimizations would result in 12 and 16 steps per second on the low end, and 24 and 48 steps per second on the high end for particle systems of 1000 volume and surface particles respectively.

7.3.6 Discussion

The equations of motion for a dynamically coupled particle system result in two sets of coupled second order non-linear differential equations (3.1) and (4.1). To determine the position and orientations of the particles at new time values, we need to solve the initial value problem for these equations. To do this, we rewrite the second order system of differential equations as sets of coupled first order differential equations and then numerically integrate the first order equations through time using a finite difference based scheme.

Implicit or explicit integration schemes can be used to integrate over a given time step. When the system dependencies can be written in matrix form, the matrix will be large, unstructured, and sparse. In addition, the non-zero values of the matrix will be continually changing making optimizations more difficult. High order explicit schemes, such as Runge-Kutta (Press et al., 1988) or semi-implicit methods (Terzopoulos et al., 1987) could be used, and for typical dynamic systems would result in good convergence and large time steps, but at the expense of a complicated implementation and possibly slow interactive response. When considering these alternatives we must keep in mind that the time step will be limited by the natural frequency of the system and that the system in question is highly correlated and oscillatory in nature. For the case of a damped non-linear oscillator, the explicit leap-frog scheme, with stable time step, is more economical than the implicit Euler scheme while yielding almost identical results (Greenspan, 1973). For most n-body calculations, simple second-order accurate explicit schemes, such as Leapfrog, provide the best compromise between accuracy, stability, and efficiency (Hockney and Eastwood, 1988). To maintain interactive update rates and simplicity of implementation, we have used both the explicit Euler and Leapfrog integration schemes. Empirical results correspond with analytical derivations showing the Leapfrog method to be superior.

[7] The exact results were 5940 pairs per step, 18,978 pairs per second, and 3.19 steps per second.

7.4 Visualization

7.4.1 Rendering

Rendering, as we know from the ray-tracing and radiosity literature, can be very time consuming. In general, high quality rendering is based on determining the reflection of light off of a continuous surface with associated material attributes. In Chapter 5 we discussed the generation of continuous surface descriptions and the rendering of those surfaces.

However we need not assume that a continuous description surface is necessary for high quality renderings. There are many cases when a surface description is not necessary or desirable when rendering an object's shape. If the object being modeled is gas-like or of an amorphous nature without definite boundaries, then the particle system can be rendered directly, thus bypassing the surface description phase. For example, Reeves (1983b) and Sims (1990) have rendered particle systems as *light emitting points* to simulate fire, water falls, and mist. Alternatively if one has assigned a *density field* to each particle, one can integrate over the density fields as Stam (1995) has done to create visually complex scenes of clouds, smoke, and fire. For rendering similar types of fluid-like behavior, 4D texture maps could also be used to interesting effect. Since the positions of an oriented particle model provide a uniform surface sampling of the object, particle systems lend themselves very well to rendering with *paint strokes* (Meier, 1996) resulting in a "painterly" rendering style.

7.4.2 Visual Cues for Real-Time Use

For interactive modeling, we prefer rendering techniques which maximize our understanding of the system. Except for the final stages of an animation, it makes more sense to spend the computing power on providing informative displays for the user instead of regenerating complete surface descriptions. The display should convey information we are interested in such as particle position, orientation, energy, neighbor connections, and provide quick approximations of the final surface. Such real-time visual cues are indispensable in debugging, scripting animations, and modeling. Here we review some techniques we have found useful.

Light Emitting Points

Rendering each particle as light emitting points is the simplest and quickest method. The result is an uncluttered scene in which all of the particles are visible. X-Y position information is obvious and depth perception is enhanced by rotating the scene in real-time. Additional information can be encoded in the color of the particle. For example the heat energy of a particle is easily displayed as a variation from white (cold) to red (hot).

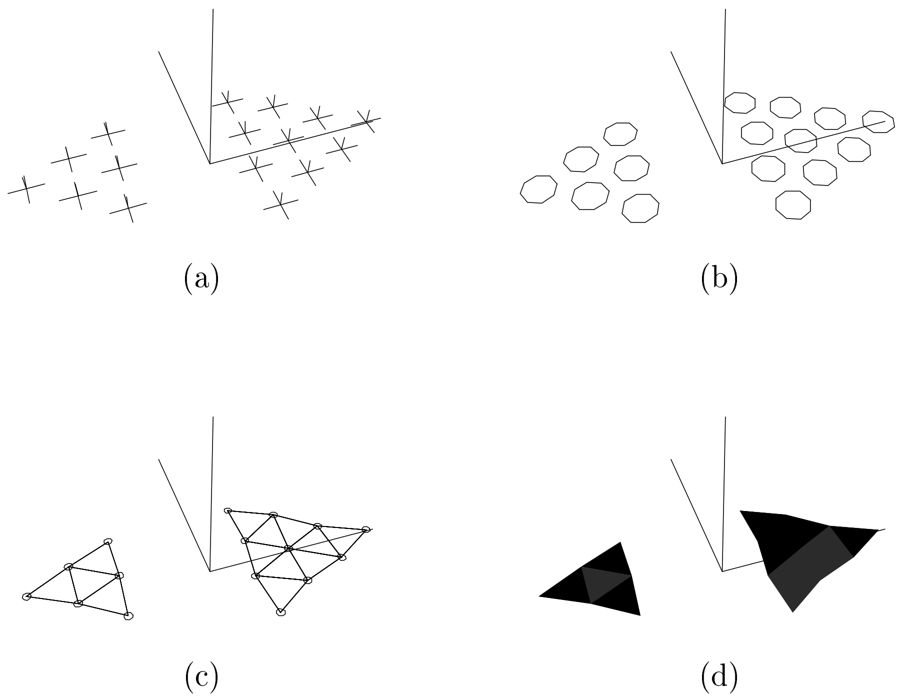


Figure 7.10: Visualization techniques

(a) axes, (b) discs, (c) wireframe triangulation (d) flat-shaded triangulation.

Wire Frame Particles

Wire frame representations of a particle are useful for approximating the volume or surface area represented by a particle without obscuring the other particles. For a solid modeling particle system, we draw the particle as a star: three mutually perpendicular line segments. Oriented particles can be drawn with the positive Z axis highlighted to indicate orientation. We have found displaying a wire frame hexagon for each oriented particle results in an intuitive feel of the surface while still conveying orientations (Figure 7.10).

Surface Approximations

To quickly approximate the surface of a solid model, the particles can be displayed as spheres or cubes. Another simple approximation of the iso-surface is to render a sphere for each particle and shade the sphere according to the gradient of the summed fields at that point (Miller and Pearce, 1989). To approximate the surface resulting from oriented particles, the particles can be displayed as filled hexagons. Displaying the neighbor connections between particles as line segments is another method for quickly approximating the surface while at the same time conveying the structure of the particle system. A slightly longer process is to display a wire triangulation of the surface (Figure 7.10). The triangulation process is given in Section 5.2.3.

7.5 Summary

Our dynamically coupled particle system presents several computational challenges. In particular, the continually changing spatial relationships between particles complicates a variety of tasks. The computation of inter-particle forces, particle creation heuristics, and the triangulation algorithm are defined over all particle pairs making these $O(N^2)$ problems that must be solved at each time step. Our solution is to convert the problems from global problems involving all N particles, to local problems involving at most a constant number of nearby particles. This effectively transforms the problem to $O(N)$ time plus the time needed to find the neighboring particles.

The nearby neighbors problem, or the range search problem, is the problem of finding all particles within a given distance from a point in space. It computes the spatial relationships between points, suggesting we should organize the data according to the embedding space of the points. By using spatial subdivision techniques, which allow one to focus on relevant subsets of data, the range search problem can be solved on average in time $O(N)$ time for volume modeling using a fixed grid, and $O(N \log N)$ time for surface modeling using a region-based tree structure. To reduce computation during a simulation, we cache the results of neighbor computations for several time steps.

In order to compute the state of the system at new time values, we first calculate the inter-particle and external forces. To compute the forces in linear time, we limit the inter-particle force functions to a fixed distance bound, ignoring distant particles. We use a weighting function to decay our potential energy fields to zero at a fixed distance. In addition to reducing force computations, this approach provides a correct energy based representation of the splitting and merging of objects.

The differential equations of motion describe how a system of particles will respond to inter-particle and external forces. Equations (3.1) and (4.1) are second order nonlinear differential equations, making them difficult to integrate analytically. Instead, numerical integration techniques, based on a Taylor series expansion, are used to approximate the solution by taking finite steps through time. To do this, we reduce the second order set of differential equations of motion to two first order equations and solve the initial value problem forward in time. We have used simple explicit integration methods rather than more complicated implicit methods. We have empirically compared two low order explicit schemes. We also analyzed the leap frog method to derive analytic equations of stability and accuracy. Our empirical results match well with theoretical analysis. More sophisticated numerical high order explicit schemes, semi-implicit, and implicit schemes could also be used but it is unclear whether they would be fast enough for interactive applications. Physicists studying n-body problems have asserted that simple 2nd order time-centered explicit schemes work as well or better than low order implicit schemes, thus supporting our decision.

For interactive rendering of state we have suggested a variety of rendering styles (Section 7.4). A discussion of generating continuous surface descriptions is given in Chapter 5.

Through spatial subdivision of space and the use of explicit integration schemes, we have been able to reduce the total computational complexity of our system to expected computation times of (N) for volumes and $O(N \log N)$ for surfaces.