Programmatically Typed and Coupled Soft Body Multi-scope Particle Modeling

Justin Dierking

Abstract

I am demonstrating a method of particle based simulation in which the interactions between particles are programmed to produce complex behaviors. The purpose of such complexity is to produce soft body models that can be deformed, torn, and execute phase changes as a result of temperature change. The behavior of a soft body model is controlled by defining a series of particle types and defining relationships between particle types. These relationships use bilinear interpolation to generate coefficients from displacement and temperature to control thermal loss, normal acceleration, static drag, dampening, repulsion, attraction, and heating.

Categories and Subjects: Governing Mathematic Principles – Sampling – Computation – Simulations – Deformation and Tearing – Phase Change

Introduction

The purpose of this project is to achieve a working understanding of efficient, single threaded design for enumerating complex interactive, soft body particle systems through the use of a computer. These soft body particle systems are to be capable of modeling the effects of thermal loss, normal acceleration, static drag, dampening, repulsion, attraction, and heating. To realize this, software was developed in the languages of C++ and Python.

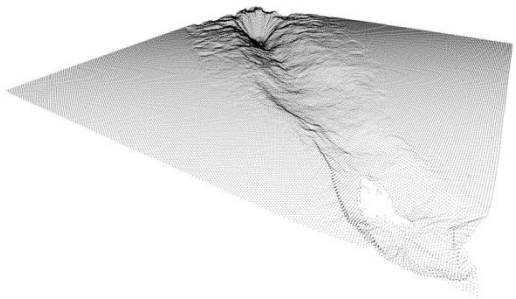

Figure 1: *Simulation of a collision of a projectile fired at an oblique angle and a deformable surface. This simulation was composed of 67000 particles. Two types of particles were defined for the projectile and the surface respectively and relationships defined for the two types of particles.*

This paper is intended to provide the reader with an understanding of the development and operation of this program. Specifically, governing mathematic principles, program design, and execution of a simulation are discussed.

Governing Mathematic Principles

This project is based on the application of a second order Taylor expansion of the Euler method. The process of running a simulation for a fixed unit of time can described as a series of explicit integrations computed with variable predetermined time steps.

The equations shown in figure 2 and 3 are applied successively to every particle for each computational pass to the integrate positions and temperatures of every particle.

$$p(t_0 + h) = p(t_0) + hv(t_0) + \frac{1}{2}h^2a^2(t_0)$$

Figure 2: *Second order Taylor expansion of the Euler method where t0 is the initial time, h is the time step, p is position, v is velocity, and a is acceleration.*

$$w(t_0 + h) = w(t_0) + hq(t_0)$$

Figure 3: *Euler method where t0 is the initial time, h is the time step, w is temperature, and q is the heat transfer rate.*

For each pass, the acceleration and heat transfer rates are accumulated. These summations are accumulated from the comparisons. The accelerations and heat transfer rates that are

accumulated are derived from functions that use displacement, average temperature, and particle type to index and compute interpolations from pre-programmed bilinear interpolation matrices.

$$a_j = \sum_{i=1}^{I} a(\|p_i - p_j\|, \frac{w_i + w_j}{2}, c_i, c_j)$$

Figure 4: *Summation where accelerations are accumulated from the acceleration function 'a' that takes the inputs: magnitude of positions 'pi' and 'pj', the average temperature of 'wi' and 'wj', the sampled particle type 'ci', and the affected particle type 'cj'.*

$$q_j = \sum_{i=1}^{I} q(\|p_i - p_j\|, \frac{w_i + w_j}{2}, c_i, c_j)(w_i - w_j)$$

Figure 5: *Summation where the heat transfer rate is accumulated from the difference in temperature multiplied by the heat transfer coefficient function 'q' that takes the inputs: magnitude of positions 'pi' and 'pj', the average temperature of 'wi' and 'wj', the sampled particle type 'ci', and the affected particle type 'cj'.*

As are result of using explicit methods to solve ODEs for position and temperature, truncation error is introduced into a simulation with each computational pass. Truncation error is proportional to a particle's velocity, acceleration, and heat transfer. If truncation error is left unmitigated, high energy simulations will become unstable and cause soft body structures to disintegrate.

To control truncation error, the time steps used for solving the particles' positions and temperatures is not constant. Each computational pass is given its own time step to use. The time step is based on the pass's accumulated velocities, accelerations, and heat transfers.

Each time step is the product of an inverse magnitude of a heat transfer, velocity, or acceleration and a precision constant. For each pass the magnitudes of acceleration, velocity, and heat transfer are maximized. The resulting time steps for acceleration, velocity, and heat transfer are minimized to produce a single time step that will be used to solve for position and temperature for the computational pass. This has the effect of normalizing the truncation error to within the precision constant.

$$h_a = \frac{P}{\max\{\|a_1\|, \|a_2\|, \dots, \|a_n\|\}}$$

$$h_v = \frac{P}{\max\{\|v_1\|, \|v_2\|, \dots, \|v_n\|\}}$$

$$h_q = \frac{P}{\max\{|q_1|, |q_2|, \dots, |q_n|\}}$$

$$h = \min\{h_a, h_v, h_q\}$$

Figure 6: *Equations used to calculate a time step 'h'. 'P' is the desired precision. ha is the inverse of the highest magnitude of acceleration. 'hv' is the inverse of the highest magnitude of velocity. 'hq' is the inverse of the highest magnitude of heat transfer. 'h' is the minima of 'ha', 'hv', and 'hq'.*

As shown in the summation functions for acceleration and heat transfer, displacement and average temperature are supplied as inputs. These inputs are used for selecting interpolated values from bilinear interpolation matrices. Each interpolation matrix is mapped to one or more combinations of sampled and affected particle type relationships. The interpolated values provided are used as coefficients for controlling dampening, repulsion, attraction, and heat transfer.

Each interpolation matrix is created with its own intervals for interpolating temperature and displacement. If the supplied displacement or temperature fall outside of the intervals of the matrix, a zero is returned. If they're within the matrix's intervals, an interpolated value is returned.

$$b_{d,t} = (i_d)(B_{10}) + (1 - i_d)(B_{00})$$

$$b_{d,t+1} = (i_d)(B_{11}) + (1 - i_d)(B_{01})$$

$$b = (i_t)(b_{d,t+1}) + (1 - i_t)(b_{d,t})$$

Figure 7: *Bilinear interpolation of a 2D matrix 'B' containing 4 data points where 'b' is an interpolated value.*

The coefficients generated by the bilinear interpolation, particle type definitions, and type

relationships are used to control the application of static and dynamic stress functions on a particle. The outputs of these stress functions are the heat transfer rates and accelerations that a particle accumulates during a computational pass.

In this project, the stress functions are hard-coded into the program. The static stresses are linked to a particle's type and behave independently of displacement or temperature. These stresses are static heat loss, normal acceleration, and static drag. The dynamic stresses are linked by relationships defined between particle types and behave dependently on displacement and temperature. These stresses are dampening, repulsion, attraction, and heat transfer.

$$q = -wc$$

Figure 8: *Static heat loss where the heat transfer rate 'q' is equal to the product of temperature 'w' and a negative coefficient 'c'.*

$$a = -vc$$

Figure 9: *Static drag where the acceleration 'a' is equal to the product of velocity 'v' and a negative coefficient 'c'.*

$$a = c$$

Figure 10: *Normal acceleration.*

$$a = c\frac{p_j - p_i}{\|p_j - p_i\|}(v_j - v_i)$$

Figure 11: *Dampening where acceleration 'a' is equal to the product of coefficient 'c', the unit vector of displacement between two particles 'p', and the difference in velocities between two particles.*

$$a = c\frac{p_j - p_i}{\|p_j - p_i\|}$$

Figure 12: *Attraction or repulsion where acceleration 'a' is equal to the product of coefficient 'c' and the unit vector of displacement between two particles.*

$$q = c(w_j - w_i)$$

Figure 13: *Heat transfer rate 'q' is equal to the product of coefficient 'c' and the difference in temperature between two particles.*

Executing a simulation based on the mathematic functions and equations described takes place sequentially.

First, heating and acceleration is accumulated for each particle using the summation equations in figures 4 and 5. Heat transfer and acceleration values are generated with the bilinear interpolation method in figure 8 combined with the dynamic stress equations. Finally heat transfer and acceleration are added using the static stress equations.

Second, with heating and acceleration accumulated, a time step can be calculated. The time step is calculated by using the inverse magnitudes of heating, velocity, and acceleration as shown in figure 6.

Third, with a time step calculated and acceleration and heating accumulated, the simulation can be integrated. Using the explicit integration functions in figures 2 and 3, the particles positions and temperatures are integrated.

Throughout the process of simulation, these steps are repeated during each computational pass. Computational passes are continuously executed until the specified amount of simulation time as been processed.

Sampling

The most time consuming step of each computational pass is the sampling of neighboring particles to accumulate acceleration and heating. Fundamentally, the behavior of two interacting particles is derived from a comparison of positions, temperatures, and the particles' types. Static and dynamic stress functions generate component acceleration and heating.

To obtain acceleration and heating from stress functions, coefficients for all static stress functions and selected dynamic stress functions have to be resolved. The coefficients for static

stress functions are statically defined in a particle's type definition. The coefficients for dynamic stress functions are the result of one or more bilinear interpolations from predefined matrices of displacement and average temperature.

To control which matrix is used for an interpolation, an indexing table is used to map matrices to dynamic stress functions based on the logical conditions of sampled particle type, affected particle type, and dynamic stress functions. A sampled particle type is a condition that is used to select which type of particle can be sampled. An affected particle type is a condition that is used to select which type of particle can be affected. In effect, particle types, dynamic stress, and bilinear interpolation matrices are linked together to form a relationship.

Complex behaviors can be achieved by combining or chaining relationships together across multiple particle types. This allows relationships between different types of particles to be customized. Relationships can be unidirectional, bidirectional, or reflexive.

Computation

The implementation of this project is based around a meshed, computational approach. Every simulation is distributed across an array sectors that act as individual computation units. Each sector has its local number system and contains and processes a small segment of the simulation. Large numbers of sectors are synchronized and linked together to form the entirety of a simulation. This implementation yields benefits that allow a simulation to host up to several hundred thousand particles.

Fundamentally, each sector is running an N-body algorithm to sample acceleration and heat transfer. The computational cost of adding more and more particles to a sector grows at a quadratic rate. As a result, sectors are kept as small as possible. The largest displacement interval in any bilateral interpolation matrix determines the smallest size a sector can be.

As indicated, simulations are composed of a large number of sectors linked together forming a mesh. Sectors are arranged in a three dimensional grid representing the entire number space of the simulation. During the sampling phase of a computational pass, the particle displacements and temperatures of the sector's adjacent sectors are sampled in addition to its own. A sector can have up to 26 adjacent sectors surrounding it.

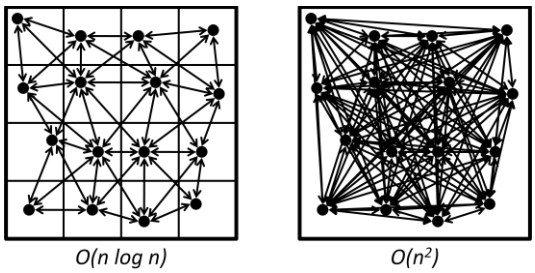

Figure 14: *Left: A depiction of sampling between particles who reside, evenly distributed, in a 4x4x1 mesh of sectors. Right: A depiction of sampling in a single sector. Each arrow terminated line represents a comparison from which displacement and average temperature are sampled. Note the difference in computational cost.*

This approach has the benefit of realizing a drastically reduced computational cost. The cost of sampling displacement and temperature from more and more particles grows at a linearithmic rate rather than a quadratic rate.

Simulations

Each simulation is built and executed using a collection of python scripts, a mesh configuration file, and finally the MPMS (Multi-scope Particle Model Simulator) executable that processes the simulation. The python scripts are used to generate initial simulation data and inject data into actively running simulations. The mesh configuration file defines the parameters of the simulation. These parameters include mesh dimensions, sector dimensions, particle type definitions, particle type relationships, and bilinear interpolation matrices.

When MPMS is executed, switches can be fed to it that alters its operation. The integration precision can be changed. The beginning and ending frames can be specified which override which frames will be processed. This allows simulations to be abruptly stopped, resumed, and interrupted with new particle data.

```
[cobra@clab mpms1]$ ./mpms_g__ --help

Usage: mpms_gcc [OPTION]...
  --help       Displays this help and exit.

Optional arguments for frame indexing and time step precision.
  -b frame     Set the beginning frame number; overrides
               default frame number of 0.
  -e frame     Set the ending frame number; overrides number of
               frames in defined in mesh.
  -t precision Set the time step precision.

Exit status:
0  if OK,
1  if minor problems
2  if serious trouble
```

Figure 15: *Screen shot of the MPMS help page.*

When MPMS is started, it initializes the simulation from the data in the mesh configuration file and populates the simulation from the particle data generated from with the python scripts. A mesh configuration file's purpose is to provide information needed to initialize a simulation. The content of the file is divided into four blocks. The first block consists of a single line that specifies the dimensions of the simulation, the number of frames to integrate, and number of CPU threads to use.

```
[sectors x] [... y] [... z] [sector width] [frames] [threads]
[number of types]
[static drag] [normal x] [... y] [... z] [static flag] [r] [g] [b] [static heat loss]
[number of relationships]
[affected type] [sampled type] [matrix id] [stress function id]
[number of matrices]
[# d data points] [d origin] [d step] [# t data points] [t origin] [t step]
[matrix data]
```

Figure 16: *Formatting of the mesh configuration file. The first and third blocks have been highlighted. The second and fourth blocks are not.*

The second block consists of the particle type definitions for the simulation. The block is prepended with the number of type definition lines. Each successive line is a type definition. A type definition consists of constants for static drag, normal acceleration, display color, and static heat loss.

The third block consists of particle type relationships. Like the previous block, this block is prepended with number of relationship lines. Each successive line consists of the type of particle that is going to be affected, the type of particle that will be sampled for comparison, interpolation matrix id, and the dynamic stress function that will be used.

The fourth and final block consists of matrices. The coefficients that dynamic stress functions use are assigned with the interpolated values made from these matrices. Each matrix is a two dimensional matrix. The axes are for average temperature and displacement. This block is prepended with the number of matrices. The successive lines are populated with the matrices. Each individual matrix is itself prepended with a header line that defines its dimensions and intervals. The data points of each the matrix follow the header line.

The python scripts and the MPMS executable use the same format to read and write particle data. A particle data file begins with a line indicating the number of particles in the file. The following lines are the particle data. Each line of data consists of the sector the particle will be added to, the particle's position inside that sector, velocity, type, time-to-live, and temperature.

```
[number of particles]
[sector x] [... y] [... z] [pos x] [... y] [... z] [vel x] [... y] [... z] [type] [TTL] [temp]
```

Figure 17: *Formatting of a particle data file.*

Deformation and Tearing

This simulation was ran to demonstrate tearing and deformation of a projectile colliding with a deformable surface. Two types of particles were defined. Four interpolation matrices were built to generate coefficients for repulsion, attraction, and dampening. Five relationships were built to govern the behavior of the particles. The temperature of all particles was kept at zero for the duration of the simulation. This simulation is composed of 67 thousand particles.

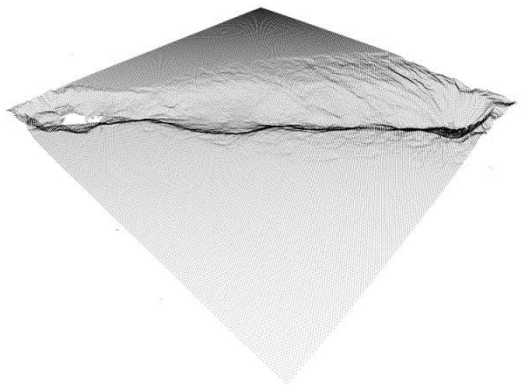

Figure 18: *Side view of the collision at frame 299 of the simulation.*

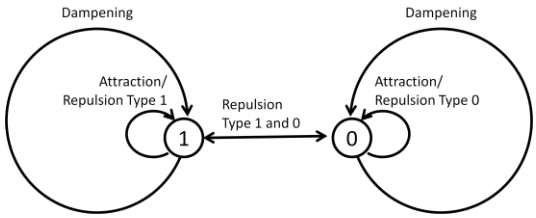

Figure 19: *Diagram showing reflexive and bidirectional relationships for dampening, attraction, and repulsion for type 0 and type 1 particles in the simulation.*

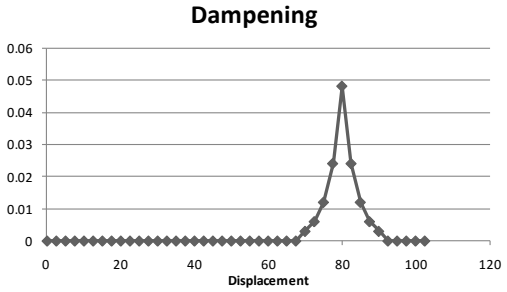

Figure 20: *Series used for generating dampening coefficients as a function of displacement. This matrix was used for both reflexive dampening relationships for type 0 and 1 particles.*

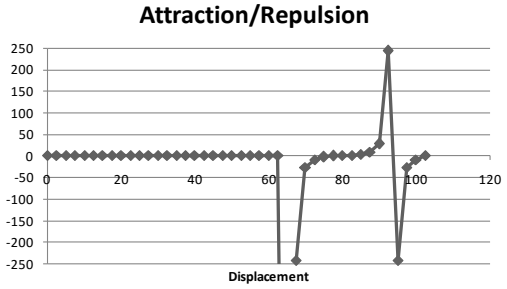

Figure 21: *Series used for generating attraction and repulsion coefficients as a function of displacement. This matrix was used for a reflexive relationship for type 0 particles. A negative coefficient will accelerate two particles away from each other, while a positive coefficient will accelerate two particles towards each other.*

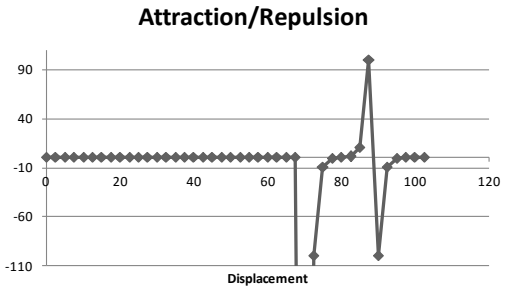

Figure 22: *Series used for generating attraction and repulsion coefficients as a function of displacement. This matrix was used for a reflexive relationship for type 1 particles.*

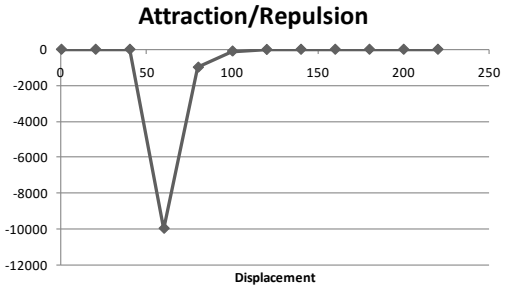

Figure 23: *Series used for generating repulsion coefficients as a function of displacement. This matrix was used for a bidirectional relationship between type 0 and type 1 particles.*

The two soft bodies in this simulation were programmed to behave in a very similar fashion. The relationships for dampening, attraction, and repulsion were designed to hold and arrest particles at a displacement of 80. This formed destructible bonds between the particles. When the soft bodies for both particles were initially generated, they were placed at equal-distant positions with displacements of 80. As stresses were undergone due to the collision, particles pushed and pulled against each other causing tearing and deformation. At the point of impact, tension between particles exceeded the bonds' attraction and a tear was formed; breaking the bonds.

Phase Change

This simulation was built to demonstrate heating and a resulting change in a soft body's behavior. In this simulation, a jet of particles is emitted at a rectangular prism resting on a substrate. As the prism's particles are heated, their temperatures rise and the bonds holding them together weaken causing the prism to collapse, flex, and melt.

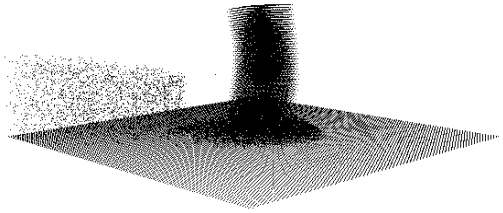

Figure 24: *Image of the melting and flexing rectangular prism resting on a substrate.*

The simulation is comprised of 46 thousand particles. These include three types of particles: type 0, 1, and 2. The prism is comprised of type 0 particles. The substrate is comprised of type 1 particles. The heating jet is comprised of type 2 particles.

The type 0 particles making up the rectangular prism are the most dynamic in the simulation. This particle type has reflexive relationships for attraction, repulsion, dampening, and heating. Dampening is proportional to temperature as shown in figure 26. The attraction/repulsion wedge shown in figure 27 is inversely proportional to temperature.

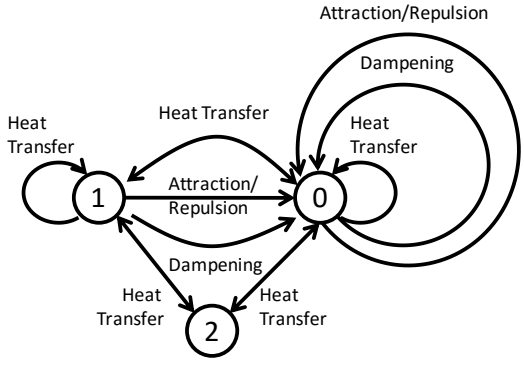

Figure 25: *Diagram showing reflexive, unidirectional, and bidirectional relationships for dampening, attraction, repulsion, and heating for type 0, 1, and 2 particles in the simulation.*

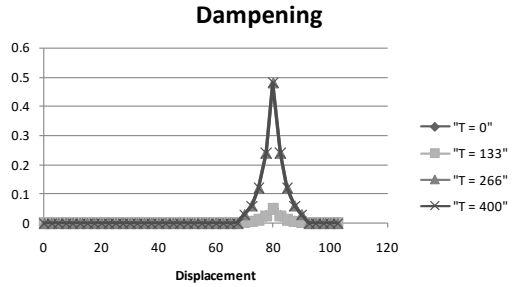

Figure 26: *2D matrix used for generating dampening coefficients as a function of displacement and temperature. This matrix was used for the reflexive dampening relationship for type 0 and the unidirectional dampening relationship sampling type 1 particles and affecting type 0 particles. In effect, as the temperature rises, the dampening coefficient at a displacement of 80 and the neighboring displacements increases.*

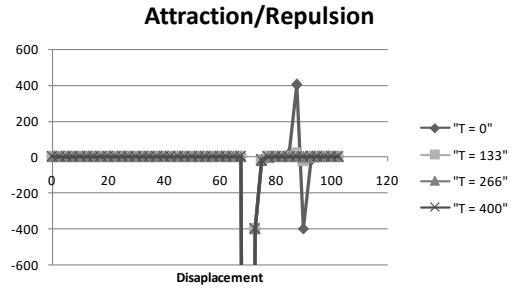

Figure 27: *2D matrix used for generating attraction and repulsion coefficients as a function of displacement and temperature. This matrix was used for a reflexive relationship for type 0 particles and the unidirectional relationship sampling type 1 particles and affecting type 0 particles. As the temperature closes to 400, the attraction and repulsion wedge closes to 0. In effect, bonds governed by this matrix weaken as the temperature increases.*

This simulation uses a simple pair of matrices for transferring heat between particles. The heating coefficients for all relationships are flat for a displacement ranging from 0 to 120. The coefficients for bidirectional heat transfer relationships to type 2 particles are .5. All other heat transfer coefficients are .01.

Conclusion

I have presented a method of particle based simulation in which the interactions between particles were programmed to produce complex

behaviors. The purpose of such complexity was to produce soft body models that can be deformed, torn, and execute phase changes as a result of temperature change. The behavior of the soft body model was controlled by defining a series of particle types and defining relationships between particle types. These relationships use bilinear interpolation to generate coefficients from displacement and temperature to control thermal loss, normal acceleration, static drag, dampening, repulsion, attraction, and heating.