©Copyright 2013 DigiPen Institute of Technology and DigiPen (USA) Corporation. All rights reserved.

Survey of Multi-Physics: A Comparison Between Coupling and Embedding

BY

Joshua Davis

BS RTIS, DigiPen Institute of Technology, Winter 2010

*THESIS*

Submitted in partial fulfillment of the requirements

for the degree of Master of Science

in the graduate studies program

of DigiPen Institute Of Technology

Redmond, Washington

United States of America

Spring

2013

Thesis Advisor: Dr. Erik Mohrmann

DigiPen Institute Of Technology

Graduate study program

Defense of thesis

The undersigned verify that the final oral defense of the

master of science thesis of Joshua Davis

has been successfully completed on May 29th, 2013

TITLE of thesis: Survey of multi-physics: A comparison between coupling and embedding

Major filed of study: Computer Science.

Committee:

Dr. Erik Mohrmann, Chair Dr. Charles Duba

Dr. Xin Li Dr. Jason Hanson

Dr. Pushpak Karnick

Approved :

Dr. Dimitri Volper date Dr. Xin Li date

Graduate Program Director Dean of Faculty

Samir Abou-Samra date Claude Comair date

Department Chair of Computer Science President

The material presented within this document does not necessarily reflect the opinion of the Committee, the Graduate Study Program, or DigiPen Institute Of Technology.

INSTITUTE of DigiPen Institute Of Technology

Program of Master’s degree

*Thesis approval*

*DATE: \_\_\_\_\_\_\_\_\_* May 29th, 2013*\_\_\_\_\_\_\_\_\_\_\_\_\_*

Based on the CANDIDATE’S successful oral defense, it is recommended that the thesis prepared by

Joshua Davis

ENTITLED

Survey of Multi-Physics: A Comparison Between Coupling and Embedding

Be accepted in partial fulfillment of the requirements for the degree of master of computer science from the program of Master’s degree at DigiPen Institute Of Technology.

Dr. Erik Mohrmann

Thesis Advisory Committee Chair

Dr. Dimitri Volper

Director of Graduate Study Program

Dr. Xin Li

Dean of Faculty

The material presented within this document does not necessarily reflect the opinion of the Committee, the Graduate Study Program, or DigiPen Institute Of Technology.

[Notes and Notation: iii](#_Toc358125225)

[Chapter 1: Introduction 1](#_Toc358125226)

[1. Multi-Physics at a Glance 1](#_Toc358125227)

[Chapter 2: Rigid-body Simulations 3](#_Toc358125228)

[1. Newtonian Impulses 4](#_Toc358125229)

[2. Constraints 7](#_Toc358125230)

[Chapter 3: Deformable Bodies 13](#_Toc358125231)

[1. Mass-Aggregates 13](#_Toc358125232)

[1.1 Spring System 13](#_Toc358125233)

[1.2 Jakobsen Constraint System 15](#_Toc358125234)

[2. Continuum Models 17](#_Toc358125235)

[2.1 Finite Element Method 17](#_Toc358125236)

[2.1.1 FEM Formulation 18](#_Toc358125237)

[2.1.2 Speed issues from non-linear strain 23](#_Toc358125238)

[2.1.3 Integration 26](#_Toc358125239)

[2.1.4 Inverted Elements 26](#_Toc358125240)

[2.1.5 Fracture and Plasticity 27](#_Toc358125241)

[2.1.6 Later Works 29](#_Toc358125242)

[2.2 Boundary Element Method 30](#_Toc358125243)

[Chapter 4: Multi-Physics 31](#_Toc358125244)

[1. Coupling 32](#_Toc358125245)

[1.1 One-way Coupling 32](#_Toc358125246)

[1.2 Two-way Coupling 33](#_Toc358125247)

[1.3 Hybrid Coupling 35](#_Toc358125248)

[2. Embedding 36](#_Toc358125249)

[3. Comparison 37](#_Toc358125250)

[Chapter 5: Implementation 39](#_Toc358125251)

[1. Newtonian Impulses 39](#_Toc358125252)

[2. Constraints 40](#_Toc358125253)

[3. Jakobsen 41](#_Toc358125254)

[4. Spring Mass Aggregates 42](#_Toc358125255)

[5. Finite Element Method 42](#_Toc358125256)

[6. Coupling 43](#_Toc358125257)

[7. Embedding 43](#_Toc358125258)

[Chapter 6: Conclusion 45](#_Toc358125259)

[References 46](#_Toc358125260)

Notes and Notation:

* The normal is always defined from object 1 to object 2, e.g. in a collision.
* Matrices are define row column format. e.g. element is row and column .
* Matrices are bolded, e.g.
* Position vectors for vertices are denoted as: . Element vectors, which are groupings of four position vectors, are denoted as: . Body vectors, which are groupings for all the vertex vectors, are denoted as: .
* Global vs. Local Indexes in FEM. A mesh is made up of N vertices and M elements. The indices are numbered for each mesh from to . Each tetrahedron contains a reference to four vertices labeled to , these are an element's local indices.
* A super-script dot indicates a total time derivative, i.e.

Chapter 1: Introduction

Most real-time physics engines focus on rigid-body dynamics because of the simple nature of the dynamic equations for rigid bodies. As computers become more powerful, there is greater ability to model complex physical interactions, such as deformable objects. If not modeled physically a large number of animations may be required to look plausible. Even then, the user cannot interact with them dynamically.

Many real-time physics engines contain deformable objects in the form of cloth. Other applications of physically based deformation have not seen widespread use yet. One reason is that the dynamic equations of deformable objects are more complex than those of a rigid body. However, these equations and formulas have been documented well enough that motivated developers can implement them. The key remaining issue preventing widespread use is the interaction between the rigid and deformable pieces of an engine.

This paper will explore some initial issues with modeling the interactions between rigid and deformable bodies. It will then cover some basic techniques of modeling rigid bodies, followed by a survey of several different deformable techniques. Finally, this paper will close with a discussion of “multi-physics”, i.e. how to model interactions between two different physical models, and implementations of some of these techniques. Examples and applications will focus on the interaction of rigid and deformable objects.

1. Multi-Physics at a Glance

A simulation that has rigid and deformable objects must address several issues in allowing them to interact. The primary reason is the many simplifications used in rigid-body engines. For instance, applied forces result in a uniform and instantaneous change to the velocity throughout the object. No matter how complex a mesh, a rigid body only has six degrees of freedom and can be modeled with a relatively small number of parameters such as center of mass, velocity, etc….

Stability is also an issue. A key feature of a rigid-body engine is the ability to stack objects. Stacking shows that the engine can model a set of complex interactions between many objects stably.

Deformable engines cannot use the assumptions that a rigid-body engine uses. Applying a force at any point on the body will not result in an instantaneous velocity change to the center of mass. The force must propagate through the interior of the body, which may take multiple frames. This is more complex, as internal interactions must be dealt with. It is significantly more difficult to obtain stable interactions, such as stacking, because all of the internal elements must be simulated.

Because of these issues, multi-physics requires significant alterations to an engine. It is important to understand how each type of simulation operates individually before discussing how to make them interact.

Chapter 2: Rigid-body Simulations

Before determining how collision is resolved, continuous versus discrete simulation must be discussed. A continuous simulation steps time forward until the time of the next collision, resolves that collision, and then continues. A discrete simulation moves all objects forward by the time step and then evaluates collisions and resolves them simultaneously afterwards. The continuous solution is more accurate, but suffers from speed issues. A set of resting objects may continue to push against each other and require a timestep of zero seconds to solve, such as circular stack of dominoes. Also, there is no analytical solution for detecting collision between rotating objects. To accurately determine when collision happens, a simulation must search for the time of collision with repeated collision-detection tests. Most modern real-time engines use a form of discrete simulation.  
 Using a discrete simulation creates the problem of objects overlapping each other after taking a time step. In addition to resolving the velocities of the colliding objects, the overlapping volumes must be addressed. This is usually referred to as correcting the penetration.

One of the earlier techniques to resolve penetration was a penalty-based approach. A penalty-based approach inserts a spring to force objects out by the penetration distance in a certain amount of time. Larger penetrations produce stiffer springs. This produces artifacts, such as penetration depth dependent restitution.

Most engines currently use an impulse-based technique instead of forces to resolve collision. The two most prevalent techniques are Newtonian and Lagrangian (constraint-based) solvers.

1. Newtonian Impulses

Newtonian impulses were first simulated by Mirtich et al [[Mirtich94]](#Mirtich94). Newtonian resolution finds the magnitude of the impulse necessary to resolve the relative velocity of the objects in the direction of the normal. That is:

|  |  |
| --- | --- |
| (1) |  |
|  |
|  |
|  |  |

where

|  |  |
| --- | --- |
|  |  |
|  |  |

and is the vector from the center of mass of object to the point of contact. It is important to note that if the velocities are separating, i.e. , then the objects are moving away from each other and should not undergo resolution. If the separating-velocity check is left out the objects will be forced back towards each other.

Using the above equations, the velocities in a collision are resolved pair wise. To solve all objects simultaneously, a system of equations must be solved. The standard approach is to iterate through all object pairs a fixed number of times. This is equivalent to a Gauss-Seidel solver with a fixed number of iterations. Resolving the velocities will not remove object penetration. To remove penetration either the velocity or position must be corrected. An extra velocity term can be added, similar to a penalty method, to push the objects apart. More typically, a simple linear projection step is used:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

where is the penetration distance. This will move the objects apart based upon their relative masses.

Following these steps will produce a reasonable simulation, but will not properly solve multi-body systems. This results in jitter, among other issues. A few simple improvements can be added in the penetration removal step. First, a slop factor can be added to only remove penetrations that exceed a certain threshold. Second, a percentage of the penetration can be resolved each frame, reducing oscillation. In combination this yields:

|  |  |
| --- | --- |
|  |  |

Even adding these features, a simulation will not be able to achieve stacking because of energy added from penetration resolution.

One problem encountered by Mirtich et al. was the modeling of resting contacts as collisions. Objects would spend time in a ballistic phase and bounce when they should rest. To fix this interactions between objects were classified as collisions or contacts based on a velocity threshold. Contacts were resolved as completely elastic collisions to prevent a loss of energy in the collision from leading to increased penetration.

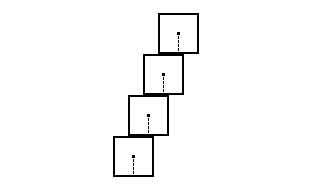
Guendelman et al. presented a method that separated collisions from contacts via a novel integration scheme [[Guendelman03]](#Guendelman03):

1. Detect Collisions
2. Resolve Collisions
3. Integrate Velocities
4. Resolve Contacts
5. Integrate Position.

By integrating velocity in-between collision and contact resolution, objects that collide will not enter contact resolution, as their velocities will be separating during the contact phase. Also, objects resting on the ground will not bounce because their velocity is resolved to zero during the contact phase before their position is integrated.

To increase stack convergence, a contact graph was used to resolve collisions and contacts starting with those in contact with static objects and moving away. Contacts iterated with a restitution of where is the total number of iterations and is the current iteration. Negative restitution does not resolve the collision, but gently slows down an object while propagating momentum transfer throughout the system.

This technique is not guaranteed to converge in a reasonable number of iterations, so shock propagation was used to enforce desired convergence. Shock propagation treats the bottom object of a pair as infinitely massive, and then resolves the contact. This step forces numerical stability at the cost of physical accuracy. While this method allowed stacking, it can produce overly stable stacks in some situations. Most notably it introduced the staircase problem shown in Figure 1 for low numbers of contact iterations. Because of shock propagation, as long as any pair would not fall, a tower would not fall.



|  |
| --- |
| Figure 1: A tower of blocks where each block's center of mass is over the one below. The tower as a whole should fall, but does not. |

Another use of contact graphs, for both speed and stability, is to "sleep" objects. If an object has not moved for a period of time, it is ignored during the update loop; i.e. not checked for collision against other objects. The simulation still checks non-asleep objects against asleep ones, but asleep objects do not check against static or other asleep objects. This optimization forcibly stabilizes a resting stack, however care must be taken when waking objects up again. A waking object must check its neighbors and wake any that are resting on it. If this is not done, then an object may stay asleep and remain floating in midair.

Newtonian solvers are simple and easy to implement. Achieving stable results requires extra complexities, each with its own issues. An additional system must be implemented to model joint motion, limits and motors.

2. Constraints  
 The constraint based system of Catto is derived from Lagrangian mechanics [[Catto05]](#Catto05). In Lagrangian systems equations are formulated in terms of potential and kinetic energy and are solved by minimizing the action required.

A constraint is a scalar equation that may be in terms of objects’ positions, velocities and time and is set equal to zero:

|  |  |
| --- | --- |
| (2) | . |

For example, is a distance constraint that restricts the length between two points on two bodies.

Position constraints are usually not solved directly but are commonly used to find and solve the velocity constraint, that is the derivative of the position constraint with respect to time. Solving the velocity constraint will restrict velocity so that the position constraint is not violated. Due to the chain rule [[ChainRule]](#ChainRule), the velocity constraint can be written in the form of:

|  |  |
| --- | --- |
| (3) |  |

where and is a scalar that adds energy into the system. is the Jacobian matrix expressed as. and are 3D vectors that describe the linear and angular bases on which the constraint is solved.

The Jacobian describes the axis that requires the least action to solve the constraint and can be generically computed for a given constraint type. Generally, the Jacobian is found by taking the position constraint, differentiating with respect to time, then determining it from inspection.

The constraint described so far is an equality constraint, but inequality constraints can also be solved with minor code modifications:

|  |  |
| --- | --- |
|  | if( <= 0)  Enforce >= 0  else  Skip |

Most notable of inequality constraints is a contact constraint. A contact constraint forces object penetration to be less than or equal to zero. A contact constraint is modeled as

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

and by inspection the Jacobian is found to be

|  |  |
| --- | --- |
|  | . |

Catto presented a technique called sequential impulses to solve for the new velocities in a constraint [[Catto05]](#Catto05). Sequential impulses is mathematically equivalent to a Projected Gauss-Seidel solver. The impulse magnitude can be solved for with the equation:

|  |  |
| --- | --- |
| (4) |  |

where is the effective mass of the constraint

|  |  |
| --- | --- |
| (5) |  |

In equation (5) is the mass matrix:

|  |  |
| --- | --- |
|  |  |

is the inverse inertia tensor and is the inverse mass matrix:

|  |  |
| --- | --- |
|  |  |

A key feature presented by Catto is accumulating and clamping the total impulse. Implemented naively, a sequential impulse would be clamped between its limits:. This does not allow applying negative impulses to remove energy that will not reverse the total momentum. The correct solution is to clamp the total impulse:

|  |  |
| --- | --- |
| (6) | . |

Numerical inaccuracies in floating point and truncation error in integration can lead to position error in a constraint. Adding energy to the constraint to remove position error is a feasible solution with proper clamping. Baumgarte stabilization is commonly used to correct position drift. The position error is fed back into the velocity constraint and adds energy into the system to push the objects back to their constrained positions. Utilizing the term in the velocity constraint to generate exponential decay towards gives:

|  |  |
| --- | --- |
|  |  |

As mentioned in [Chapter 2 Section 1](#Impulses), solving for zero penetration can produce instabilities such as jitter. A slop factor can also be added to constraints to reduce this instability. This can be extended to all constraint types. The term adds energy into the constraint, so it can also be used to model restitution. Restitution in collisions can be modeled by adding a portion of the separating velocity back into the constraint. Adding restitution when objects are resting on each other would cause instabilities, so this is commonly added only when the separating velocity is above some threshold. All together, position drift and restitution can be added to a contact constraint as:

|  |  |
| --- | --- |
|  | =Max(-slop,0) |
|  | if() |
|  | += |

Other constraint types would keep the slop factor for position drift, but not the restitution term.

Another key feature presented by Catto is warm starting. Using sequential impulses may take a large number of iterations to converge to a global solution. Most iterative solvers use an initial guess when solving. A good initial guess is the result from the last frame, especially if there is temporal coherence. Warm starting causes the solver to converge faster, allowing a smaller number of iterations to be used. To warm start with contact constraints the contact points must be cached. However, contact points come and go rapidly and applying last frame's result at the wrong point can cause instabilities. New points that are within close proximity to last frame's points use the old point's total impulse to warm start.

One of the benefits of a constraint framework is that it can model joints, motors, and limits for any joint type within the same framework that models rigid body collisions. A limit is just a restricted range on the constraint being solved, that is:

|  |  |
| --- | --- |
|  | if |
|  | Don’t Solve |

The accumulation of the total impulse in equation (6) must also be updated to deal with limits. and are set to or respectively depending on if the lower or upper limit is violated. Position drift is also altered slightly. Normally position drift is resolved with Baumgarte correction by adding to . With a limited constraint changes to either or depending on which side is violated.

Motors are modeled by adding energy to the constraint through the term. A motor is commonly defined from a velocity constraint. For example, a motor that forces two objects to rotate about the x-axis with a relative speed of 5 rad/sec is:

|  |  |
| --- | --- |
|  |  |

From inspection, the Jacobian and can be determined:

|  |  |
| --- | --- |
|  |  |
|  |  |

Real motors are not capable of infinite strength, so the min and max impulse in equation (6) are set for the motor's strength.

Constraints can be made soft, so as to behave like a spring and allow deviation from equilibrium, through the addition of the constraint softness term gamma () as presented by Catto [[Catto11]](#Catto11). The constraint softness is defined as

|  |  |
| --- | --- |
|  |  |

where and are the spring stiffness and damping terms respectively and will be more fully defined later. The constraint impulse is updated as:

|  |  |
| --- | --- |
|  |  |

where is the updated mass term

|  |  |
| --- | --- |
|  |  |

and is the updated bias term

|  |  |
| --- | --- |
|  | . |

The constraint softness adds a portion of the total impulse back into the constraint to soften how quickly the constraint is solved. Catto also showed that solving a soft constraint with semi-implicit Euler is mathematically equivalent to solving a spring with fully-implicit Euler and is therefore unconditionally stable. The spring stiffness and damping ( and ) are unintuitive values because their units include the mass of the load on the spring. They can be re-defined more intuitively as and where is the angular frequency, is the damping coefficient and is the effective mass of the constraint as defined in equation (5). Note, this is not the same as . Taking this further, can be expressed more intuitively as where is the angular frequency in Hertz. This allows more intuitive control of a spring through as the response speed in seconds and as the damping ratio where zero is no damping and one is critical damping. Discussion of the original implementation can be found on the Bullet forums [[Catto07]](#Catto07).

Constraints provide a unified framework for modeling joints, motors and limits. Stacks can also become stable with a small number of iterations by utilizing the warm starting technique presented by Catto.

Chapter 3: Deformable Bodies

Deformable bodies have been simulated for the past several decades. Applications include cloth, muscle tissue and other soft bodies. There are two main techniques for modeling deformable bodies: mass-aggregate models and continuum models.

1. Mass-Aggregates

A mass-aggregate is one of the simpler ways to model a deformable object. It is simple both in terms of implementation and of understanding. These techniques tend to be less accurate than their continuum counterparts. Mass-aggregates often suffer from having non-intuitive measurements of material properties. Despite these disadvantages, they tend to be significantly faster.

There are many different ways to implement a mass-aggregate system. The two methods that will be presented are a physically based spring system and a non-physically based constraint system.

1.1 Spring System

Spring systems have been used for several decades in computer science simulations. Applications have included modeling faces [[Terzopoulos90]](#Terzopoulos90), cloth [[Breen94]](#Breen94) [[Bridson02]](#Bridson02), soft bodies [[Matyka04a]](#Matyka04a) and volumetric spring systems [[Baraff92]](#Baraff92).

A spring system is formed by approximating a body as a collection of node points and then connecting these node points by springs. These springs enforce the shape of the object. The force for a spring is given by Hooke's Law:

|  |  |
| --- | --- |
| (7) |  |

where is the spring coefficient and measures the incompressibility of the spring and is the current displacement from the rest length. More generally:

|  |  |
| --- | --- |
| (8) |  |

where

|  |  |
| --- | --- |
| (9) |  |

and is the rest length of the spring. This is a perfectly elastic spring which would oscillate forever. To model the effect of friction within the spring, a damping force is added:

|  |  |
| --- | --- |
| (10) |  |

where is the damping coefficient.

The simplest way to integrate springs is with explicit Euler integration. That is, iterate over all springs summing up the forces, then update position and velocity:

|  |  |
| --- | --- |
| (11) |  |
|  |

A simple change that can yield more stability is to use an implicit integration for the position update, leading to semi-implicit Euler:

|  |  |
| --- | --- |
| (11) |  |
|  |

This general form of springs that is used in most games has not advanced much over the years. The primary advancement has been the use of more advanced integration techniques to reduce error. A large problem with most spring systems is that as the springs get stiffer, the simulation must take smaller and smaller time steps. If the time step is not small compared to the frequency of the spring, it will explode violently. This phenomena is commonly called the "stiff problem." Semi-implicit Euler will help this, but is not unconditionally stable. Fully implicit Euler is defined as

|  |  |
| --- | --- |
| (12) |  |
|  |

and is unconditionally stable but requires the future position and velocity which cannot always be solved for. Many authors have proposed integration techniques that are implicit or allow larger time steps but most of these come at a large computational expense [[Chen98]](#Chen98), [[Debunne99]](#Debunne99).

The other primary advancement in the field of spring systems is to rework the system to a generalized particle system. This particle system is solved for in terms of the energy required to solve constraints and springs are generally replaced with soft constraints. Some examples can be found in [[Breen94]](#Breen94) and [[Baraff92]](#Baraff92).

An interesting idea was presented by Matyka et al. to model soft bodies [[Matyka04a]](#Matyka04a) using a virtual gas inside the body to apply pressure. A pressure-model soft body consists of a spring system that has a pressure force applied to each triangle on the face of the body. This keeps the body inflated while only having surface springs. This technique can be quite useful in games because of its simplicity and speed, however it has a very limited usage because only "inflated" objects can be modeled with it. Matyka later released more implementation details about pressure-model soft bodies [[Matyka04b]](#Matyka04b).

As stated earlier, one of the largest issues with a spring system is that modeling stiff bodies is more difficult. There are more advanced integration techniques, but they don't solve all problems. Because of these issues, a non-physically based mass-aggregate system will be explored in the next section.

1.2 Jakobsen Constraint System

Jakobsen used a mass-aggregate constraint system for the game *Hitman: Codename 47* [[Jakobsen03]](#Jakobsen03). The focus of this implementation was to create a stable and fast system where both deformable and rigid objects could be modeled. The heart of the technique was to use Verlet integration for stability. In this system, everything is turned into a point-mass system constrained at the position level. Position is modified directly; no velocity constraint is solved using the Jacobian. This works well within a Verlet integrator as it is expressed only in terms of position, acceleration and previous position. Stated formally, Verlet is

|  |  |
| --- | --- |
| (13) |  |
| . |

This allows directly modifying position to satisfy a constraint which will correct the behavior of the particles.

A simple distance preserving constraint can be modeled by moving the particles to the correct position.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

This will correct the two particles to be at the constrained distance, acting like a very stiff spring. To make the constraints soft, a variable can be added to control what fraction of the error is corrected each frame.

Collision resolution is easily modeled by clamping a particle to the surface of what it hits. This automatically fixes the velocity as if restitution were zero. To add restitution the old position is simply reflected across the collision plane and then scaled by the restitution coefficient.

To solve multiple constraints, the simulation iterates over all constraints sequentially which is equivalent to Jacobi or Gauss-Seidel iteration. Softness can be added by giving each constraint a maximum number of iterations to be solved. For example, plants were only allowed one iteration making them soft and flexible.

Rigid bodies can be modeled by adding constraints to a collection of point masses until they have only 6 degrees of freedom. Articulated bodies can be modeled by connecting particles together with fewer constraints, allowing degrees of freedom to remain, such as pin or hinge joints. To make advanced interactions, angular constraints need to be modeled. Angles can be enforced with a few techniques. The first is to have a distance constraint to another part of the body that only activates upon certain conditions (inequality constraint). For example, a constraint can be added between the knees of a ragdoll to enforce a minimum distance so the legs don't cross. Angle constraints can also be modeled with dot products:

|  |  |
| --- | --- |
| (14) |  |

While it is simple to determine if this constraint is violated, it is not simple to determine how the offending particles should be moved. No further details were presented by Jakobsen about solving a constraint with equation (14).

Friction can also be modeled in this system, but the presented method is not a robust solution. By fixing the old position along the direction of motion the velocity will be reduced. Care must be taken in doing this to not reverse the direction of motion. Friction is one of the less elegant parts of this technique.

One of the other downfalls of this algorithm is that it is not physically based. While it does allow visually pleasing collisions, it leads to difficultly in stacking as well as other complex behaviors. Since there is no velocity it can be more difficult for designers to create game logic with physics.

2. Continuum Models

Mass-aggregate systems will always suffer from approximating a body as a set of discrete points. The mass of the objects is lumped into the nodal points and the volume is preserve with springs. Instead, the full continuum of points can be evaluated by approximating a volume as a series of elements. These continuum methods are more expensive than their mass-aggregate counterparts, but are more accurate. Two notable continuum methods are Finite Element Method and Boundary Element Method.

2.1 Finite Element Method

Finite Element Method (FEM) is one of the most popular methods of modeling a continuum model in Computer Sciences. The pioneering work in moving from spring-mass cloth simulations to FEM was made by Terzopoulos et al. [[Terzopoulos88]](#Terzopoulos88). In its simplest form, FEM uses a volumetric mesh of tetrahedra to approximate the continuum of an object. Other element types can be used instead of tetrahedra, but due to their complexity they will not be addressed here. For a parallel discussion of FEM see [[Erleben05]](#Erleben05). Note the differences in notation.

2.1.1 FEM Formulation  
It is import to first discuss the notation used in FEM. Each object consists of M elements and N vertices. Each element is a tetrahedron that contains references to four vertices. For each vertex, an undeformed position and a deformed position are stored. The current displacement of any element can then be denoted as

|  |  |
| --- | --- |
| (15) | . |

The displacements from all nodes are grouped together into one vector for the body

|  |  |
| --- | --- |
| (16) | . |

Each node also contains a force and velocity vector which are grouped together with the same notation.

To model deformable bodies the generalized form of Hooke's Law,

|  |  |
| --- | --- |
| (17) |  |

is discretized with FEM where is defined in equation (16), is the nodal force vector and is the stiffness matrix. Solving equation (17) is a Linear Complementarity Problem (LCP). An LCP can be solved with an LCP solver, which solves equations of the general form:

|  |  |
| --- | --- |
| (18) |  |

where is a known matrix, is a known vector, and is an unknown vector. As the dimension of increases, solving for the inverse of can be both computationally unfeasible and numerically unstable. Instead, LCP solvers find directly without inverting . There are numerous LCP solvers, including Jacobi iteration and Gauss-Seidel, but the standard choice for FEM is the Conjugate Gradient Method, which is covered spectacularly by Shewchuk [[Shewchuk94]](#Shewchuk94).

In equation (17), is the global stiffness matrix and is the nodal force vector. The unknown being solved for is the nodal displacement vector . The strain relation of each element needs to be determined so that the stress can be calculated each frame and used to generate .

First note that any displacement within a tetrahedron can be represented as:

|  |  |
| --- | --- |
|  |  |

where is the barycentric coordinate for the point within the tetrahedron. This can also be written as:

|  |  |
| --- | --- |
|  |  |

where . is the shape function and linearly interpolates any vector function from the nodes to a point within the tetrahedron.

An individual element's stress can be computed with linear Cauchy Strain, which is a linearized Green's strain. Linearizing raises issues that will be addressed in [Chapter 3, Section 2.1.2](#StiffnessWarpingSection). Linear Cauchy Strain is defined as

|  |  |
| --- | --- |
| (19) |  |

where the 's and 's represent axes, e.g. 0, 1 and 2 represent the x, y and z axes respectively. Linear Cauchy Strain yields a symmetric matrix with only six independent values and can be expressed as

|  |  |
| --- | --- |
|  |  |

Expanding and then pulling out gives:

|  |  |
| --- | --- |
|  | . |

The strain for a point within the tetrahedron can be represented as a linear interpolation:

|  |  |
| --- | --- |
|  |  |

Evaluating the structure of we can see that:

|  |  |
| --- | --- |
|  | . |

Further expanding one of these items:

|  |  |
| --- | --- |
|  |  |

where , and .

The partials within can be computed as follows: first let

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

and

|  |  |
| --- | --- |
|  | . |

Each partial for nodes 1 through 3 can then be defined as:

|  |  |
| --- | --- |
|  |  |

and

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

via conservation of momentum.

Stress can be related to strain by

|  |  |
| --- | --- |
| (20) |  |

where is the 6x6 isotropic elastic matrix that is defined once for the whole body. is defined using Young's modulus () and Poisson's ratio ():

|  |  |
| --- | --- |
| (21) | . |

For static equilibrium the work done on the object at its surface by all forces equals the work done in its interior by stress [[Erleben05]](#Erleben05).This is the work balance equation:

|  |  |
| --- | --- |
| (22) | . |

In this equation, is the work performed by internal stress forces. is the work performed by the external nodal forces. is the work performed by the external load. Solving for the internal work gives which yields the element stiffness matrix:

|  |  |
| --- | --- |
| (23) |  |

where is the volume of the element. The individual element stiffness matrices are combined into a global 3nx3n stiffness matrix. In practice, the global matrix is never actually assembled, as it is sparse. Instead, non-zero elements are stored and accessed during solving.

To efficiently assemble the global stiffness matrix, the structure of must first be investigated. First note that is a 12x6 matrix. Now can be rewritten as:

|  |  |
| --- | --- |
|  | . |

After closer inspection, can be expressed as:

|  |  |
| --- | --- |
|  |  |

where . The global matrix can be assembled with the pseudo code:

|  |  |
| --- | --- |
|  | for all tetrahedra  for in [0,3]  for in [0,3]  = globalIndex()  = globalIndex()  += |

Note that for isotropic materials is symmetric, that is .

After assembling the global stiffness matrix, the new positions can be computed using equation (17). This is the elostatic solution which does not account for inertia, damping forces or external forces. Adding these terms yields the equation of motion:

|  |  |
| --- | --- |
| (24) |  |

Grouping like terms and expanding known terms yields:

|  |  |
| --- | --- |
| (25) | . |

is the total mass matrix which can be diagonalized without significant error by turning it into what is called a lumped mass matrix. can be written as

|  |  |
| --- | --- |
|  |  |

where is the sum of all the mass terms for each element that node is in. An individual element's mass can be computed with

|  |  |
| --- | --- |
| (26) |  |

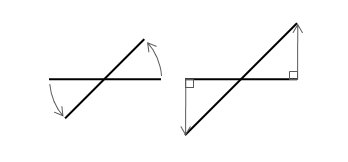
where is the density of the object, is the undeformbed volume of the element and is the 3x3 identity matrix. As for the remaining term in equation (25), is the Rayleigh damping matrix

|  |  |
| --- | --- |
| (27) |  |

where and are the mass damping and stiffness damping coefficients. Common values are Hertz and seconds so that is based solely on and is therefore diagonal.  
 Once all of these terms are known it is possible to solve Equation (25). However, this is rarely done. In most simulations the new position is integrated for. Equation (25) is used to solve for the new velocity. After discussing some advancements in the field of FEM, a semi-implicit Euler integration scheme will be discussed in [Chapter 3, Section 2.1.3](#FemIntegration).

2.1.2 Speed issues from non-linear strain

While FEM has existed since the late 80s, it has not seen widespread use in real-time simulations. The primary issue was that the Green-Strain tensor is not linear and strain changes each time step. A non-linear strain is also more difficult to solve both from a computational and a numerical stability standpoint. Most early real-time simulations tried to get around this non-linear strain somehow. The first attempt was to approximate the strain with a linear strain tensor. Linear strain is constant for a mesh and therefore only has to be computed once. Linearizing strain results in errors from the linear approximation, most notably during large deformations as shown in Figure 2. These errors cause a mesh to inflate unnaturally during a rotation. Because of these errors, speed improvements were sought out in other areas.



|  |
| --- |
| Figure 2: Normal rotation (left) and linearized rotation (right). The error due to linearized rotation increases with the size of the rotation. |

One of the earliest real-time applications was made for simulating muscle tissue [[Debunne99]](#Debunne99). In order to run in real-time, a level of detail system was created to modify the coarseness of the mesh dynamically. A high detail mesh was used in areas undergoing large deformations while other areas used a coarse mesh. Also, Lamè equations were used for solving FEM instead of the typical large matrices. For further speed improvements, a topological octree was used to store the parent/child relationship of meshes. Further adaptations were made to fix boundary conditions [[Debunne00]](#Debunne00). In [[Debunne01]](#Debunne01), the Lamè equations were replaced with explicit FEM. Explicit FEM does not solve the global system. It calculates the forces for all the nodes of an individual element and applies them in isolation. A non-linear Green-strain tensor was used.

Around the same time, Bro-Nielsen et al. used a technique called condensing for a surgery application [[Bro-Nielsen96]](#BroNielsen96) that only modeled surface nodes. Condensing does not ignore the internal elements, but computes the effect that the internal elements have on the surface nodes. Condensing is numerically expensive but can be pre-computed in an offline step. Many additional offline computations were taken to speed up the simulation. Unfortunately, topological changes cannot be modeled without having to re-compute the offline values. Without topological changes fracture cannot be modeled.

Muller et al. tried to use linear strain and remove the rotational artifacts with a new technique called stiffness warping [[Muller02]](#Muller02). By calculating the forces in a non-rotated frame, many artifacts of linearization are removed. These forces are transformed to the rotated frame and then. The rotation was factored out per node and produced quick and stable results without standard linearizing artifacts. However, total momentum is not conserved. Therefore, ghost forces were produced.

Later, Muller et al. proposed an alteration to stiffness warping [[Muller04]](#Muller04). Instead of factoring rotation per node, it was factored per element. No ghost forces were produced with this modification. Using stiffness warping, no offline techniques were needed to operate in real-time. Stiffness warping is now a standard process in most real-time applications of FEM. Simulations that are focused on accuracy over speed, such as medical and engineering tools, still use a non-linearized strain. A derivation of stiffness warping follows.

First note from equations (15) and (17) that an individual tetrahedron has the elastic force:

|  |  |
| --- | --- |
| (28) | , |

which is expanded to

|  |  |
| --- | --- |
| (29) |  |

and rewritten as

|  |  |
| --- | --- |
| (30) |  |

where is the internal force of the element in its undeformed state. Applying stiffness warping, where is the rotation matrix that rotates each node in the tetrahedron into , leads to

|  |  |
| --- | --- |
| (31) |  |

which expands to

|  |  |
| --- | --- |
| (32) |  |

This can be rewritten as:

|  |  |
| --- | --- |
| (33) |  |

where and .   
 This solution assumes that the rotation from an undeformed element to a deformed element is known. The rotation can be found simply by letting and then finding the matrix:

|  |  |
| --- | --- |
| (34) | , |

where and . The rotation matrix can be extracted from in a number of ways. The simplest is to Gram–Schmidt orthonormalize and then take the result as .

2.1.3 Integration

With the information from stiffness warping, the final integration equations for FEM can be assembled. Note: all vectors are body vectors. Using equation (25) and implicit discretization, and are updated from timestep to :

|  |  |
| --- | --- |
| (35) | . |

Using 1st order explicit derivatives and integration to rewrite and

, this becomes:

|  |  |
| --- | --- |
| (36) | . |

Expanding and collecting terms gives

|  |  |
| --- | --- |
| (37) |  |

where . Now the LCP can be solved where and . This is the velocity step of a semi-implicit Euler integration. Position is updated by .

2.1.4 Inverted Elements

An issue with the above formulation of FEM is that if an element inverts, it will stay inverted. One approach is to add extra stiffness forces to prevent inversion, but this is not practical as inversion can happen during a high impulse collision, giving forces no time to prevent the inversion. Instead, gracefully recovering from inversion can be implemented.

Teschner et al. changed the FEM formulation to add a volume preservation term that penalizes inversion [[Teschner04]](#Teschner04). Nesme et al. showed simpler approaches to enforce equation (34) to be right-handed [[Nesme05]](#Nesme05). The simplest is called QR decomposition and checks left-handedness by making sure that the last vertex is on the correct side of the triangle formed by the other three vertices. If it is on the other side, then the sign of the last basis axis is flipped, making the basis right handed. This will produce the correct effect if the inverted vertex was not a member of the triangle used in the QR decomposition. If any other vertex is inverted then the element is restored by a large rotation. While this might seem like a large problem, in large meshes it produces no visual artifacts. After this, the resulting matrix is run through a Gram–Schmidt orthonormalization to produce the final rotation matrix.

The second method to fix inversion presented by Nesme et al. is to perform a polar decomposition on to find the closest orthogonal frame. With this method, the inversion is detected by a matrix determinant and then finding the offending vertex and correcting its axis's sign. This method is used when more accurate handling of inverted elements is desired.

Irving et al. presented a fully robust method to recover from inverted and degenerate elements [[Irving04]](#Irving04). This formulation is quite different from the method outlined above. Normally Green strain is used to compute stresses and forces, but Irving et al. stopped before that step and worked all formulas out using the deformation gradient. This allowed the detection of inverted elements. While this method is robust in all cases, it is more expensive and complex.

2.1.5 Fracture and Plasticity

FEM has the advantage of being incredibly versatile. Many different behaviors can be detected and modeled with slight alterations to the formulations. One of these is the modeling of both brittle and ductile fracture.

O`Brien et al. modeled brittle fracture by measuring the strain tensor of an element [[O`Brien99]](#OBrien99). This could measure if an element should be split. If so, it would split along the eigenvector of the strain tensor. This showed that fracturing an object is conceptually simple. To remesh the object after its topology changes is a more difficult problem to solve. A technique for this was discussed by O`Brien et al.

Most objects in real life do not exhibit a pure brittle-fracture reaction, even objects typically thought of as brittle tend to deform slightly. Ductile fracture is the behavior that most objects exhibit, allowing deformation before breaking. O`Brien et al. modeled ductile fracture by adding plasticity to FEM [[O`Brien02]](#OBrien02). This model of plasticity split the total strain into elastic and plastic strain:

|  |  |
| --- | --- |
| (38) | . |

The total strain can be measured as

|  |  |
| --- | --- |
| (39) | . |

Introducing the yield material property, the plastic strain can be updated if the elastic strain exceeded the yield. Another material property was added to cap the maximum plastic strain. Later, Muller et al. added a term called plastic creep that controlled the fraction of current excess strain that was added to the total plastic strain [[Muller04]](#Muller04). Combining all of these together, the plastic strain update is:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

Plastic strain causes plastic forces that can be modeled as

|  |  |
| --- | --- |
| (40) | . |

This plastic force should be subtracted from each corresponding node's force, or alternatively it can be added to the external force vector of that node.

2.1.6 Later Works

There have not been many advancements to the FEM formulation since the survey by Nealan [[Nealan05]](#Nealan05). Since then, FEM was first used in a commercial game: *Star Wars: The Force Unleashed*.

Parker et al. presented details to the techniques used to both speed up the simulation for real-time and graphical "tricks" for added realism within the requirements of real-time [[Parker09]](#Parker09). Speed improvements came from threading and using SSE intrinsics for parallel computations. During fracture, tetrahedra were only split along pre-existing boundaries in order to maintain a consistent framerate and reduce level balancing complications. To compensate for this, splinters were added to allow extra graphical realism. Splinters are an embedded graphical mesh that controls the appearance of fracture. During fracture, the centroid of a splinter would determine which tetrahedron to stay attached to (See Figure 3).

|  |  |  |
| --- | --- | --- |
| Splinters.png | | |
| Figure 3: Detail is embedded into the geometry of the mesh. When the mesh breaks the splinters follow the tetrahedron that the centroid was inside of. |

Splinters allowed a high degree of artistic control as to how a mesh could appear when breaking\. Using this, wood would splinter along the grain and bricks would break along the mortar. While this might seem like artificially scoring the mesh for fracture, the underlying mesh is still broken according to physics calculations. The splinters are only used for extra graphical detail without letting the number of elements grow to a non-real-time level.

2.2 Boundary Element Method

Unlike most other continuum methods, the Boundary Element Method (BEM) only uses a surface mesh and not a volumetric mesh. A separate rendering mesh is not needed. BEM still analyzes the full continuum of points within the mesh even though only the surface mesh is used. The Green-Gauss theorem is used to turn volume integrals into surface integrals, turning a 3D problem into a 2D problem. Being reduced to 2D greatly reduces the complexity of the system being solved and substantial speed gains can be made.

Even with the reduction to simulating a 2D surface, BEM requires a large computational time. There are ways to achieve real-time performance, as was demonstrated by James et al. [[James99]](#James99). Many of the calculations necessary for BEM can be precomputed. Precomputing has issues with changing boundary conditions, such as contacts. James et al. used the Sherman-Morrison-Woodbury formula to update the precomputed values since only minor changes to the block matrix should happen. With this BEM was able to run at real-time rates.

There are several drawbacks to BEM. Only homogeneous materials can be modeled since only the surface mesh is stored. It does not handle topological changes well. BEM must update the global stiffness matrix whereas FEM only needs to update the local matrix. This is the primary drawback, as topological changes, e.g. fracture, are one of the key benefits of continuum models.

Chapter 4: Multi-Physics

Connecting multiple types of physical simulations together is a complex process (see [Chapter 1, Section 1](#Section1)). There are many difficult issues to solve, such as efficiency and merging simulation capabilities [[Baraff97]](#Baraff97). Commercial middleware physics engines are often used, and it is difficult to modify an existing code base to work with another system. Sometimes the code is not available to modify. Beyond this, each kind of simulation has its own techniques and assumptions used for reasons of both efficiency and accuracy. Particle systems handle collision very differently than rigid-body systems. Spring mass-aggregates and FEMs have very different methods as well. Beyond the implementation details, the underlying method of solving these systems is different, e.g. a fluid system uses different differential equations than a rigid-body system. Because of these reasons, combining physical simulations can be excessively complicated or unfeasible.

Despite these complications, many authors have modeled multiple physics systems interacting together. One of the earliest methods was presented by Baraff and Witkin with a technique called interleaving [[Baraff97]](#Baraff97). The idea behind an interleaved simulation is to alternate stepping forward between two systems instead of solving them simultaneously. Each system then uses the other system's previous result during solving. The major advantage to this type of simulation is that each system can be solved without knowledge of the other’s current solution. Only forces need to be transferred between the systems. However, interleaving results in a discrepancy between the two simulations and a repair step must be made. One major problem is that interleaving works best when there is a large mass disparity between systems. Despite the issues with interleaving, this became a base for future research into multi-physics.

The majority of techniques can be classified into one of two categories: coupling or embedding. Unfortunately, coupling and embedding are highly-overloaded terms. These terms will be more fully explained in the following sections, most similarly to the definitions used by O`Brien et al. [[O`Brien00]](#OBrien00).

1. Coupling

A coupling between systems is formed by making pair-wise interactions between two systems. These systems have no knowledge of the coupling itself. Each system defines the reactions for objects within itself. The coupling sits outside of both systems and applies forces between the two systems. For instance, in a cloth vs. rigid-body coupling, the rigid-body system will resolve rigid-body vs. rigid-body collision while the cloth system will resolve cloth vs. cloth collision. The coupling transfers the forces between rigid-bodies and cloth.

There are many different forms of coupling, but they can be broken up into three categories: one-way, two-way and hybrid couplings.

1.1 One-way Coupling

In one-way coupling there is a primary and passive system. As the primary system is often driven by some external energy source, the primary system transfers forces to the passive system while the passive system does not transfer any forces back. While this is not physically accurate, it produces reactions in the passive system that can be reasonable and visually pleasing.

A one-way coupling is useful when there is a large mass difference between the two systems. In such a case, the amount of force that the passive system can apply to the primary system is small enough that it can be ignored. A common example of this is the clothing on a person. Cloth is heavily affected by a person's movement, but the cloth has a minor effect on the person.

In a one-way coupling, the primary and passive systems can run independently of each other. The passive system only reacts to changes in the primary system. The primary system does not care about the passive system. Because of this the timestep of the two systems does not have to be the same. The primary system can run at 30 fps while the passive system can run at 300 fps for improved accuracy and precision.

Sometimes one-way coupling is used even though there should be a small visible effect of the passive system onto the primary system. For example, O`Brien et al. showed a one-way coupling between a cloth and rigid-body system to model a basketball hitting a net [[O`Brien00]](#OBrien00). While the effects of the net on the ball are small, there is a noticeable effect. In these situations a viewer may feel like something is wrong without being able to determine exactly why.

More complicated interactions can be modeled simply when one-way coupling is used. Quite often, the passive system is the one of interest and the primary system would be very complex to model realistically. In these cases it is simpler to move the primary system directly and have the passive system act in a one-way coupling. Li et al. used one-way coupling to model soil slippage and manipulation [[Li93]](#Li93). A bulldozer was moved through an environment via user controls and the soil responded to this movement. The soil did not affect the movement of the bulldozer. As the bulldozer was not the primary interest of the simulation, but rather the soil, one-way coupling was ideal.

1.2 Two-way Coupling

When the effects of the passive system on the primary system need to be modeled accurately, two-way coupling is ideal. In this case, the systems cease to be primary and passive as both transfer forces equally. Two-way coupling is more realistic. O`Brien et al. demonstrated this with real-life comparisons [[O`Brien00]](#OBrien00).

In a two-way coupled system the interaction between each pair of systems is modeled in the best method for that pair. To do this effectively, each pair of systems must sync up their timesteps. If a coupling between a rigid-body system running at 30 fps and a cloth system running at 300 fps is formed then both systems must run at 300 fps. The coupling should take place after each system has resolved its own interactions but before position is integrated. If this is not possible due to using an external code base then the coupling must take place at another time, reducing accuracy.

Coupling can be significantly more computationally expensive than running each system independently, even after accounting for the additional interaction time. O`Brien et al. presented an example of this: System B requires short timesteps where the amount of work required each timestep is small. System A does not require a short timestep, but the amount of work each timestep is large. The timestep of the total simulation is dictated by system B for stability and therefore system A must run at the shorter timestep. Even if the interaction between system A and B is simple, the total computational time spikes (see [Figure 4](#Figure3))*.* Because of this a two-way coupled system can take too long to run and become less useful, despite being more accurate.

|  |  |  |
| --- | --- | --- |
| CouplingTime.png | | |
| Figure 4: Top: running the systems separately at their native timesteps. Bottom: running each system together at the higher frequency results in a significantly larger total time. |

Since O`Brien et al. presented in 2000, the processing power of computers has increased to the point where accurate two-way coupling is feasible in real time. Sifakis et al. used two-way coupling to model the interactions between many different systems with a technique called soft binding [[Sifakis07]](#Sifakis07). A hard-bound particle was fixed to a point on the surface of an object. A soft-bound particle was connected to this hard-bound particle but was allowed to drift away. The drift of a soft-bound particle was corrected with either a spring force or by directly moving the soft-bound particle to the hard-bound particle. Collision between coupled systems was handled by applying impulses between soft-bound particles.

Shinar et al. created contact constraints between rigid-body and FEM systems in a fully-coupled system using Newmark integration [[Shinar08]](#Shinar08). The collision detection and resolution tested the vertices of the FEM against the rigid bodies and created contact constraints between points found to be colliding. This system allowed embedding rigid-bodies within deformable-bodies, e.g. the animation of a rigid-body skeleton that drove the motion of deformable muscles.

1.3 Hybrid Coupling

Two-way coupling is ideal for accurate system interactions. However, large computational costs can increase development time by increasing the time between iterations, which decreases productivity. In some cases, accuracy is not necessary, only a believable response. Hybrid couplings transfer forces between the primary and passive systems, but use an approximation that is not fully accurate. Like one-way coupling, hybrid coupling does not require an equal timestep between the systems.

Although hybrid coupling is less accurate, it has a number of advantages over two-way coupling. Making a hybrid coupling of two systems is significantly easier. A wide variety of hybrid couplings can be used, depending on accuracy, speed and development time requirements. These approximations tend to have many parameters than can be exposed for tweaking, allowing greater design control. Hybrid couplings are also computationally cheaper, allowing greater frame rates, as well as shorter turnaround time between iterations, which increases productivity.

While hybrid couplings are not as accurate as their two-way counterparts, they are more realistic than a one-way coupling. They can be used to remove the visual disturbances associated with one-way couplings. Several examples were presented in O`Brien et al. that used hybrid couplings. A hybrid coupling of a basketball hitting a net produced more realistic behavior than a one way coupling since the ball's trajectory was altered by the net.

2. Embedding

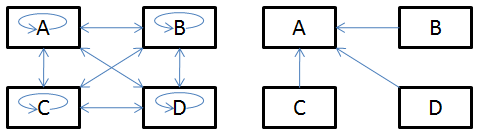
Embedding is defined as the following: one system is defined as the primary system; all other systems will define one or more proxies per object that are solved in the primary system as if they were objects native to that primary system. For example, FEMs embedded into a rigid-body framework will interact with other rigid bodies and other FEMs as if they are rigid bodies. Embedding defines the interactions for all systems within itself. If a system wants to define its internal interactions separately then the embedding framework must allow exceptions to be made.

Parker et al. embedded a rigid-body system into a deformable-body FEM system [[Parker09]](#Parker09). Each rigid-body created an FEM proxy that was extremely stiff. These proxies were then solved as if they were ordinary FEM objects. Afterwards, the forces were approximated on the center of mass and applied to the rigid-body. This was especially useful in Parker et al.'scase as a commercial rigid-body engine was used and creating a coupling between these two systems would be extremely difficult.

Fishman embedded FEM and spring-mass systems into a rigid-body framework [[Fishman12]](#Fishman12). This allowed for the benefits of rigid-body systems, such as stacking, with these systems. Although not presented by Fishman, other features such as generic constraints ([see Chapter 2, Section 2](#ConstraintsSection)) can be added onto systems since they are embedded into a rigid-body system. One downside of this is that inertia tensors need to be computed for proxies, whereas Fishman gave his rigid-body proxies infinite rotational inertia.

3. Comparison

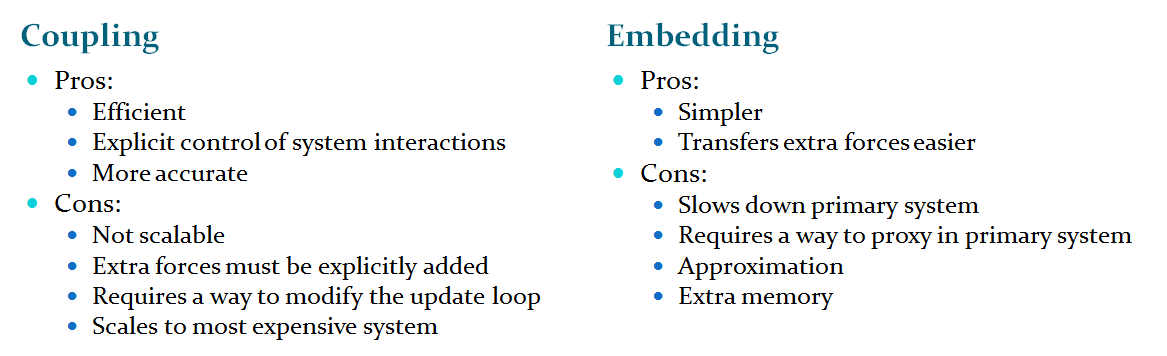
Embedding can be thought of as a form of hybrid coupling, however the key difference is that the number of interactions to be defined is greatly reduced (see Figure 5).



|  |
| --- |
| Figure 5: Left: a coupling between four systems results in ten couplings. Right: All systems embedded into A results in three proxies. |

Embedding has the advantage that creating a proxy for a system is often simpler than creating a coupling. In addition to handling collision resolution between two systems, a coupling must also handle the collision detection. A collision detection function must be created for every pairing of shapes in both of the systems. Creating a separate broadphase to reduce collision detection checks can be complicated.

A coupling must also properly transfer forces from effects other than collision. This means that the coupling must have intimate knowledge of both systems. The coupling must duplicate all extra functionality of each system which can excessively increase development time. For example, if a rigid-body system has planetary gravity that is defined by a collision sphere, this must be transferred to all FEMs within range. Embedding naturally handles transferring all forces from the primary to passive system(s). Embedding is not easily able to transfer extra forces from a passive to primary system.

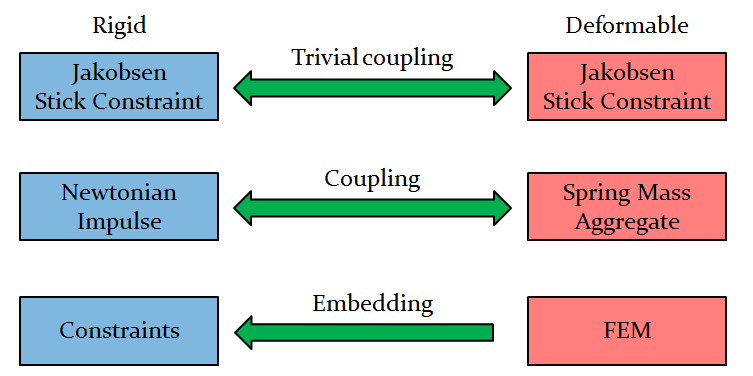


|  |
| --- |
| Figure 6: Pros and cons of coupling vs. embedding. |

Embedding also allows interactions between systems where the interaction was never explicitly defined. This is helpful in defining interactions between two systems where it might be hard to create an explicit coupling between them. For instance, if both FEM and fluid are embedded into a rigid-body system, FEM and fluid will automatically interact even though no explicit interaction was created.

Embedding is less accurate than two-way coupling. However, in many real-time simulations accuracy is not as important as quick, stable and realistic-looking animations. A downside of an embedded system is that the overall performance of the primary system will likely be reduced. In the case of the primary system being a rigid-body constraint system, the inner loop of the constraint solver must deal with letting the proxy update the underlying object. For instance, if a triangle in an FEM has a rigid-body proxy, the velocity change must be applied to all the vertices of the FEM so that the next constraint will have up-to-date values.

Chapter 5: Implementation

Three main projects were implemented to model different multi-physics simulations (see Figure 7). The first was a trivial coupling using the Jakobsen constraint system. A hybrid coupling was implemented between the Newtonian impulse and spring-mass aggregate systems. Finally, FEM deformables were embedded into a constraint system.

|  |
| --- |
| Figure 7: The three multi-physics projects implemented. |

1. Newtonian Impulses

Newtonian impulses were implemented in the style of Guendelman et al. [[Guendelman03]](#Guendelman03). The majority of the resolution was handled identically. Impulses were computed as shown in [Chapter 2, Section 1](#Impulses). The update loop used the same Newmark integration technique to separate collisions and contacts. Contacts iterated with a negative restitution just as Guendelman et al. to propagate energy. Contacts also underwent shock propogation.

Differences in implementation arose due to the focus on real time. Instead of level-set geometry, convex meshes were collided using the Separating Axis Theorem. Another key difference was that Guendelman et al. was overly aggressive in removing pairings during collision and contact solving. Instead, pairings were not removed from the iteration list. If the velocities were separating only that iteration was skipped in case later corrections at other contact points caused a non-separating velocity.

1. Constraints

Constraints were implemented almost entirely after the Sequential Impulse technique presented by Catto [[Catto05]](#Catto05). The primary differences in implementation were details in contact solving.

Catto presented solving the friction directions where one axis is aligned in the direction of the projection of the velocity onto the tangent plane. Instead of this, the project used a technique that generates persistent tangent directions for the same normal. Aligning a friction direction with the velocity projection works fine, however when warm starting the cached impulse will be applied to an axis that is likely in a different direction. To fix this persistent tangent directions were chosen to take advantage of temporal coherency. This however is a worse approximation of the friction cone and can lead to some odd behaviors when one axis enters static friction before another one.

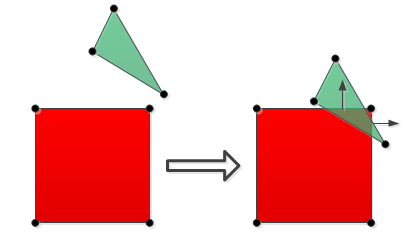
Another difference was in the max bounds for friction. Catto proposed using a fixed constant based upon gravity, the friction coeffecient and the effective mass of the constraint. This does not properly reflect the increased normal force on a box at the bottom of a tower. Instead, the actual normal force of the constraint was used for the friction bounds.

In the process of warm starting, old points must be matched to new points to properly transfer the accumulated impulse. Catto did this with contact ids based upon the clipped edges involved in creating a point in the manifold. Instead, a simple proximity check in local space was used to determine if a new point was within a certain threshold of an old point.

1. Jakobsen

Jakoben’s point-mass-aggregate system was implemented almost the exact same as presented in [Chapter 3, Section 1.2](#JakobsenSection). The key difference is how collision detection was handled. Collision was performed between the surface triangle meshes of each shape. The resulting contact point’s dynamics data was then computed from barycentric weights of the nodal points and used in resolution. As Jakobsen did not present how to detect collision, the author chose to collide surface meshes.

One major problem in colliding surface meshes is dealing with thin shell collision and generating accurate collision manifolds. To help mitigate this, only the counter-clockwise normal of a triangle is considered. It is easy for a triangle to clip an edge in such a way that it gets forced out of the side instead of the top (Figure 8).



|  |
| --- |
| Figure 8: A triangle colliding with the surface of a cube. Note on the right that the triangle gets a normal to the right which will resolve it violently to the side. |

One alternative would have been to collide points with other shapes, however this would require shape matching to turn deformable bodies into analytic shapes for detection.

1. Spring Mass Aggregates

The spring-mass-aggregate model implemented is a combination of several other author’s works. Springs are connected between the point masses using equations (8) and (10) in [Chapter 3, Section 1.1](#SpringSystemSection). These forces are integrated using a semi-implicit Euler scheme. Cloth is created using structural, shear and bend springs [[Bridson02]](#Bridson02). Generic triangle meshes are connected using graph distances to determine the cost to get from one node to another [[Erleben05]](#Erleben05).

Pressure-model-soft bodies were applied on the base of the spring-mass aggregates using the implementation details presented by Matyka et al. [[Matyka04b]](#Matyka04b). In particular, the pressure force for each triangle is computed as:

|  |  |
| --- | --- |
|  |  |

where V is the total volume of the mesh, A is the area of the triangle, is the normal of the triangle and is the pressure constant. The volume is efficiently computed using Gauss-Green’s theorem.

1. Finite Element Method

Finite Element Method (FEM) deformables were modeled as presented in [Chapter 3, Section 2.1](#FEMSection). The equations in this section were mostly taken from the text by Erleben [[Erleben05]](#Erleben05). Fracture was not modeled. Inverted elements were handled using the technique described by Nesme et al., as described in [Chapter 3, Section 2.1.4](#Section_2_1_4) [[Nesme05]](#Nesme05).

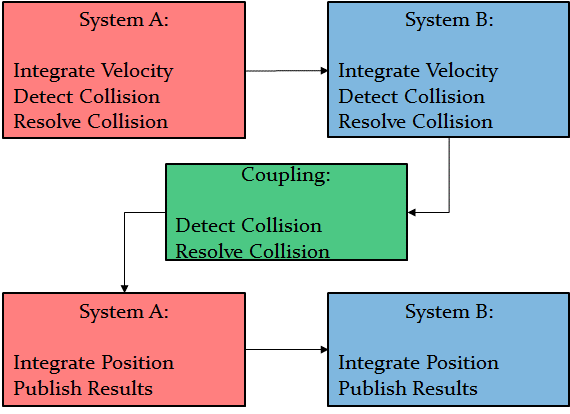
A dynamic axis-aligned bounding-box tree was used for efficiency in the collision detection algorithm. This tree was rebuilt every frame as the mesh deformed. Each object that was in contact with the FEM’s total AABB queried through all the levels of the AABB tree to determine which tetrahedra to collide with.

Tetrahedral meshes were created using the program NetGen [[NetGen]](#NetGen) .

1. Coupling

Two-way coupling was implemented most similarly to Sifakis et al. However, their work was a generalized particle-system engine. The coupling presented here is an explicit coupling between rigid bodies and spring-mass aggregates. The spring-mass aggregates collide with what is equivalent to Sifakis et al.’s hard-bound particles. These particles are used in the impulse equation (1) with the rotational terms of the particle zeroed out.

The update order of each system was interrupted to allow the coupling to affect the results for the current frame (Figure 9). Without this, the velocities would be updated at too late of a time, causing the effects of dynamic resolution to not be felt until after position had been updated to the next frame. This coupling could be made more accurate by iterating over all collisions from all the systems in one solver. Instead, each system is solved independently. This was left to a future work.



|  |
| --- |
| Figure 9: The update order for each system is modified so the coupling can affect the velocities before position integration. |

1. Embedding

The embedding work is an extension of Fishman’s implementation [[Fishman12]](#Fishman12). Fishman embedded FEMs into a constraint-based rigid body framework by turning the point of contact into a point mass for collision response. Instead, this implementation uses a full proxy for a rigid body. Rigid-body parameters are calculated as described below: center of mass, inertia tensor, linear and angular velocity.

A rotation matrix is calculated for use in constraints that need to keep track of local points or axes. This allows a full rigid body proxy for use in constraints other than contacts.

For the rigid body proxies, each tetrahedron was given the mass of the total mesh. This means each vertex was given the mass of one quarter the total mesh’s mass. One reason for this is to avoid having the mass of the mesh change as the volume changed. The total mass was used for faster convergence during constraint solving. The constraints and the FEM deformables would have to be solved in the same LCP in order to have a true coupling.

The center of mass is computed as:

|  |  |
| --- | --- |
|  |  |

where is the total mass, and and are the mass and local position of node i respectively. The inertia tensor can then be calculated using Steiner’s Theorem:

|  |  |
| --- | --- |
|  |  |

where is the original inertia tensor (zero for point masses), is the 3x3 identity matrix and is the outer product between vectors. The linear velocity at the center of mass can be computed from the total momentum:

|  |  |
| --- | --- |
|  |  |

Angular velocity is approximated from the total angular momentum:

|  |  |
| --- | --- |
|  |  |

Chapter 6: Conclusion

Multi-physics is still a very new field that has not seen much work. Commercial physics engines are starting to support features such as cloth. Other features such as fracture are becoming popular, but these are usually not solved realistically or in real-time.

A problem with more accurate models is computational efficiency. Processors are not becoming significantly faster anymore. Instead, computers are becoming multi-core. Most algorithms (LCPs) are not naturally solved with threading. To achieve real-time performance on complicated accurate models such as FEM, algorithms must be extended for multi-core support.

The industry is still heavily focused on modeling different kinds of physics systems. Current methods are already accurate enough for believability. In order to make simulations more believable these systems must be combined together. We must see a shift in focus from more accurate/efficient simulations to multi-physics. Only when this happens can simulations take the next step towards modeling our world and all of its complex interactions.

References

[Baraff92] Baraff D. and Witkin A. 1992. "Dynamic Simulation of Non-penetrating Flexible Bodies." *Computer Graphics* 26 (2): 303-308.

[Baraff97] Baraff D. and Witkin A. 1997. "Partitioned Dynamics." *Tech. Report CMU -RI-TR-97-33* (Robotics Institute, Carnegie Mellon University).

[Breen94] Breen D., House D., Wozny M. 1994. "Predicting the Drape of Woven Cloth." *SIGGRAPH.* 365-372.

[Bridson02] Bridson R., Fedkiw R., Anderson J. 2002. "Robust Treatment of Collisions, Contact and Friction for Cloth Animation." *ACM Transactions on Graphics* 21 (3): 594-603.

[Bro-Nielsen96] Bro-Nielsen M. and Cotin S. 1996. "Real-time Volumetric Deformable Models for Surgery Simulation using Finite Elements and Condensation." *Computer Graphics Forum* 15 (3): 57-66.

[Catto05] Catto, E. 2005. "Iterative Dynamics with Temporal Coherence." *Game Developers Conference.*

[Catto07] Catto, E. n.d. *Bullet Forums.* Accessed May 15, 2013. http://bulletphysics.org/Bullet/phpBB3/viewtopic.php?f=4&t=1354.

[Catto11] Catto, E. 2011. "Soft Constraints." *Game Developers Conference.*

[ChainRule]. n.d. Accessed May 15, 2013. http://en.wikipedia.org/wiki/Chain\_rule.

[Chen98] Chen Y., Zhu Q., Kaufman A. 1998. "Physically-based Animation of Volumetric Objects." *Computer Animation 98. Proceedings.* 154-160.

[Debunne99] Debunne G., Desbrun M., Barr A., Cani M. 1999. "Interactive multiresolution animation of deformable models." *Eurographics Workshop on Computer Animation and Simulation.* 133-144.

[Debunne00] Debunne G., Desbrun M., Cani M., Barr A. 2000. "Adaptive Simulation of Soft Bodies in Real-Time." *Computer Animation.* 15-20.

[Debunne01] Debunne G., Desbrun M., Cani M., Barr A. 2001. "Dynamic Real-Time Deformations using Space & Time Adaptive Sampling." *ACM SIGGRAPH.* 31-36.

[Erleben05] Erleben K., Sprorring J., Henriksen K., Dohlman H. 2005. *Physics Based Animation.* Charles River Media.

[Fishman12] Fishman, T. Summer 2012. *Real-Time Deformation with the Finite Element Method and Embedding Using Rigid Body Proxies.* Master Thesis, DigiPen Institution of Technology.

[Guendelman03] Guendelman E., Bridson R., Fedkiw R. 2003. "Nonconvex Rigid Bodies with Stacking." *SIGGRAPH.* 871-878.

[Irving04] Irving G., Teran J., Fedkiw R. 2004. "Invertible Finite Elements For Robust Simulation of Large Deformation." *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation.* 131-140.

[Jakobsen03] Jakobsen, T. 2001. "Advanced Character Physics." *Game Developers Conference.* www.gamasutra.com/view/feature/131313/advanced\_character\_physics.php.

[James99] James D. and Pai D. 1999. "ArtDefo: Accurate real time deformable objects." *SIGGRAPH.* 65-72.

[Li93] Li X. and Moshell J. M. 1993. "Modeling Soil: Realtime Dynamic Models for Soil Slippage and Manipulation." *SIGGRAPH.* New York: ACM. 361-368.

[Matyka04a] Matyka M. and Ollila M. 2003. "Pressure Model Soft Body Simulation." *SIGGRAPH.* 20-21.

[Matyka04b] Matyka, M. 2004. "How To Implement a Pressure Soft Body Model." Accessed May 15, 2013. www.matyka.pl/.

[Mirtich94] Mirtich B. and Canny J. 1994. "Impulse-based Dynamic Simulation." *Workshop on Algorithmic Foundations of Robotics (WAFR-94).*

[Mirtich95] Mirtich B. and Canny J. 1995. "Impulse-based simulation of rigid bodies." *Symposium on Interactive 3D Graphics.* Monterrey, CA.

[Muller02] Muller M., Dorsey. J, McMillan L., Jagnow R., Cutler B. 2002. "Stable Real-Time Deformations." *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation.* 49-54.

[Muller04] Muller M. and Gross M. 2004. "Interactive Virtual Materials." *GI ’04: Proceedings of Graphics Interface 2004.* 239-246.

[Nealan05] Nealan A., Muller M., Keiser R., Boxerma n E., Carlson M. 2005. "Physically Based Deformable Models in Computer Graphics." *Computer Graphics Forum 25 (4):* 809-836.

[Nesme05] Nesme M., Payan Y., Faure F. 2005. "Efficient, physically plausible finite elements." *EUROGRAPHICS.*

[NetGen] Schöberl, J. 2009. Accessed May 15, 2013. http://www.hpfem.jku.at/netgen/.

[O`Brien99] O`Brien J. and Hodgins J. 1999. "Graphical Modeling and Animation of Brittle Fracture." *Proceedings of SIGGRAPH 1999.* 287-296.

[O`Brien00] O`Brien J., Zordan V., Hodgins J. 2000. "Combining Active and Passive Simulations for Secondary Motion." *IEEE Computer Graphics and Applications* 20 (4): 86–96.

[O`Brien02] O`Brien J., Bargteil A., Hodgins J. 2002. "Graphical Modeling and Animation of Ductile Fracture." *Proceedings of SIGGRAPH 2002.* 291-294.

[Parker09] Parker E. and O’Brien F. 2009. "Real Time Deformation and Fracture in a Game Environment." *Eurographics/ACM SIGGRAPH Symposium on Computer Animation.*

[Shewchuk94] Shewchuk, J. 1994. "An Introduction to the Conjugate Gradient Method Without the Agonizing Pain." http://www.cs.cmu.edu/~jrs/jrspapers.html.

[Shinar08] Shinar T., Schroeder C., Fedkiw R. 2008. "Two-way Coupling of Rigid and Deformable Bodies." *SCA `08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics symposium on Computer animation.* 95-103.

[Sifakis07] Sifakis E., Shinar T., Irving G., Fedkiw R. 2007. "Hybrid Simulation of Deformable Solids." *ACM SIGGRAPH/Eurographics Symp. on Computer Animation.* 81-90.

[Terzopoulos88] Terzopoulos D. and Witkin A. 1988. "Physically Based Models with Rigid and Deformable Components." *IEEE Computer Graphics and Applications* 8 (6): 41-51.

[Terzopoulos90] Terzopoulos D. and Waters K. 1990. "Physically-Based Facial Modeling, Analysis, and Animation." *Journal of Visualization and Computer Animation* 1 (2): 73-80.

[Teschner04] Teschner M., Heidelberger B., Muller M., Gross M. 2004. "A Versatile and Robust Model for Geometrically Complex Deformable Solids." *Proceedings of Computer Graphics International (CGI).* 312-319.