The graphics simulation of volumetric deformable objects and their fracturing behaviors is needed in many applications. Currently such simulations are usually solved by using the Finite Element Method (FEM), which requires extensively large amounts of calculation and results in slow processing speed, so that it is usually not suitable for real-time simulations. On the other hand, the Particle Spring System (PSS) is largely used in simulating cloth and shell (infinitely thin) objects and provides a reasonable simulation speed that could achieve real-time animation. However due to its linear nature the PSS has the limitation of failing to capture the volumetric properties of the object. Also the pure rigid body simulation, which is volumetric and fast to simulate, is not deformable. In this paper we propose a hybrid modeling method called OSS to combine the Particle Spring System and Rigid Body Simulation methods to construct a volumetric object and to achieve fast deformable simulation. This includes fracturing behavior in real-time interactive graphics.

**Keywords:** Particle Spring Systems, Rigid bodies, Deformations, Fracturing, Real-time simulations

**INTRODUCTION**

**General Background**

Physical modeling and its visualization is one of the main tasks in computer graphics research today. The results and their applications are widely used in video games, the films industry, computer aided engineering design and training, e.g., for surgery and aircraft pilots. In some cases such as special effects for movies or animations, the simulation focuses on reality and visual effects rather than on performance so rendering speed is never the problem of concern in these cases. In such applications, the system is built with complicated and fine-detailed modeling methods and physical formulas (usually non-linear) that require large amounts of calculation. To achieve maximum accuracy it may take hours to generate one frame (Selle et al., 2009). On the other hand, real time simulation and visualization of the real world could be very costly and slow to generate, and the balance between realism and performance is difficult to achieve even when the
requirement of accuracy has already been reduced. The simulation of interaction between and within the objects, such as collisions, breaking, splitting or merging requires more careful consideration for optimal realism and optimal performance. In addition, interactive graphics in real time requires unconditional stability, which means that the system should be well-behaved in all circumstances. Therefore the selection of a modeling method and its corresponding time integration method are also important. A Particle Spring System (PSS) is considerably easier to implement. However it is difficult to combine linear springs so as to achieve real volumetric behavior. FEM is more accurate at representing volumetric behavior of objects but it requires more complicated calculations to analyze the model and the meshes are only suitable for small deformations. In this paper, we are only concerned with games applications and therefore we focus on performance rather than realism and accuracy, in contrast with engineering and physical simulation application fields.

This research relates to real-time simulations which require a different approach from simulations that require accuracy such as off-line simulations. Real-time simulations used for instance in visualization and video games have their own priorities. The first priority is unconditional stability. In physical simulations the system contains a large number of factors and they are governed by a set of complex equations. These equations are approximations to reality and most of them are modified to a simpler form (for example, from non-linear to linear) to suit real-time calculations. Thus from the moment that the simulation starts, the results will lose accuracy and will be different from the real world experience. Such a system is highly unstable and could be easily disturbed by unexpected causes, e.g., the over elastic problem in cloth simulation (Terzopoulos et al., 1987). With off-line simulation one can modify parameters or even manually adjust the graphics frame by frame to avoid this problem, while in real-time simulations the system will develop by itself following the way it has been set as time passes. A control method has to be added in or the equations need to be optimized to prevent the system from becoming unstable. The second priority is the accurate response of the system to user input (for interactive computer graphics). For most cases the simulation only has to give an approximate appearance of reality and these responses have to be stable as mentioned in priority 1. The third priority is speed. And the last priority is the simulation accuracy based on realism.

History

The study of deformable objects began in the 1980’s with Weil (1986) and Terzopoulos et al. (1987). In their work the researchers assumed the objects to be very simple, e.g., a 2D infinitely thin cloth object with only elastic effects considered. The results of these initial researches, which used particle spring systems, however had the appearance of simulating a rubber sheet. The over-elastic problem of the particle spring system could not be solved by simplifying the algorithm, and the properties of the system are largely dependent on how the spring meshes are constructed. Researchers developed more complex ways to simulate the 2D deformable objects. For example approaches using mesh refinement in (Zhang and Yuen, 2001) and multi-meshes in (Bridson et al., 2003) were developed. Baraff et al. (1998) made a great contribution to the integration method for large-time-step simulation. Recently researchers are
using high-power GPU and CPUs to do the simulation with multi-threading (Selle et al., 2009). In the meantime a few researchers have attempted to employ the Finite Element Method for simulating the internal structure of deformable objects in order to give a realistic representation of continuum objects, for example (Müller et al., 2004).

Besides the modeling and force analysis methods, collision between objects and the determination of response after the collision is also a difficult task. Many researches have been undertaken in this area and several solutions have been developed, including bounding-volume hierarchies, distance fields, stochastic methods, spatial subdivision (Teschner et al., 2004) and penalty methods (Moore and Wihelms, 1988). However these methods remain imperfect and impose heavy computational loads during the simulation. In this paper we present a collision detection and response method suitable for real-time animations as in video games with low impact on performance.

The Combined System

Some research work has already been done before on using combined methods to achieve semi-deformable simulation for different purposes Jansson et al. (2001) proposed a method of using a particle spring system to represent deformable parts and using rigid body rules to calculate the behavior of rigid body parts. Their approach considers rotations for the rigid body parts which we do not include as our rigid body parts are small and their motions are of short duration from one equilibrium state to another. By not requiring rotation data storage and computations we aim for faster real-time performance. Additionally Jansson et al. do not consider fracturing which is essential to our approach. Lenior and Fonteneau (2004) used the combined system to build a constraint system to simulate linked rigid body and deformable body. These authors also consider rotational mechanics of the rigid body parts which are not required in our approach.

Modelling

In our method, every deformable object would be modelled as three different meshes, as suggested by Müller et al. (2008). These three meshes are called the geometric rendering mesh, the collision detection boundary mesh and the physical structure mesh.

The geometric rendering mesh is made of triangles and it is only for rendering display purposes. It describes the fine detailed exterior structure of the object, and the texture that will be applied on this mesh. This mesh is subject to change based on the physical structure mesh described below.

The collision detection boundary mesh is a coarse structure of the geometry of the object. It is exterior to the object and contains the rendering mesh. This mesh is invisible and subject to change based on the deformation of rendering meshes.

The physical behavior of the object being modelled is given by the physical structure mesh. We analyze the physics of the object moment by moment and put the result into the physical structure mesh. Once the physical structure mesh is computed we then use it to create the geometric rendering mesh. From the geometric rendering mesh we then define the collision detection boundary mesh. The physical structure mesh itself is built from three different parts, as described below.
Sometimes, for simplifying the progress and speeding up the simulation, we can combine the above three meshes to one or two meshes if the geometry is relatively simple. For purposes of testing our algorithm we are only concerned in this paper with viewing the physical mesh in our test cases.

THE HYBRID METHOD

The Physical Modelling Layout

In this paper we present a hybrid method called OSS for deformable object simulation. To implement this method, the physical structure mesh of each object is made up of two different types of modelling methods:

Hard Parts (HP): These parts of the physical structure mesh are the parts where the object is assumed to behave as a pure rigid body. They are non-deformable, non-fracturable, and non-fragmentable and can only perform displacements. Rotation of hard parts is not required in our applications and this saves considerable processing time. These parts of the mesh are modelled as simple static meshes and this will reduce and simplify the mesh calculations during the simulation. Though the hard parts don’t deform, all the vertices of a hard part are defined as particles and the total mass of a single hard part is the sum of the masses of all its particles (vertices). The total force applied to a HP is the sum of the net forces on each vertex of the HP. The forces applied on those vertices need to be calculated in order to determine the amount of displacement of the hard part in each time step.

The purpose of HPs in the physical construction mesh is to avoid extensive and unnecessary calculations of the parts that are designed to be non-deformable. This has three advantages: (1) It saves modeling internal details such as internal vertices and the connections between these vertices within the HP in order to decrease memory usage and associated computation. (2) It saves physical analysis and simulation of the modeling details and thereby speeds up the whole progress of the simulation. (3) It defines a boundary of the deformation, e.g., the maximum depth that a dent could be after collision.

For most of our research we are using equal-sized blocks defined by 8 vertices for the HPs though in general this is not necessary and the HPs are arbitrary rigid shapes.

Soft Parts (SP): These parts of the physical structure mesh are the ones that have elastic proprieties. They are defined as springs and act as the connection between hard parts and apply forces to the HPs to form a constraint system. In our system the SPs have no particles of their own. Normally between two HPs there are several spring connections. The connections (which are the springs) only connect between the vertices of the HPs. Each spring is massless and also has a rest length and a maximum length, which is proportional to its rest length. Each spring has one ending particle on one HP, and the other ending particle on the other HP, as shown in Figure 1. Figure 1 shows 6 HPs connected by springs. When rendered with these vertices for the geometric rendering mesh the 6 squares behave as hard parts of the object and the parts of the object between the squares behave as the soft parts of the object.

As our method represents the object structure of rigid bodies connected by springs, we call the method “Object Spring System (OSS)” in order to distinguish it from a Particle Spring System (PSS).
Differences Between OSS and PSS

In our method (OSS), the HPs are represented by rigid polyhedra where each vertex is a particle with mass and the vertices of different hard parts are connected by massless and damped springs. Therefore the OSS is a generalization of the PSS where for the PSS each hard part is a single particle. OSS is different from PSS because the hard parts cannot be thought of as particles (even though they do not rotate) because of the following reasons:

1. The original purpose of this research was to find a method that can capture volumetric properties. As many researchers have pointed out (Zhang and Yuen, 2001), PSS failed to construct stable large solid continuous objects. It will only form deformable structures to a certain thickness. On the other hand our OSS model is capable of forming large objects as the majority of the object is constructed by solid rigid bodies (HPs) and only the parts which we required to be deformable and fracturable are made of springs (SPs). Currently for the ease of the problem the HPs are defined equally and have relatively small size. However there is no limitation that the HPs should be all of the same small size. In general HPs can be of any shape and any size, so as to form very complicated geometrical objects.

2. The spaces occupied by HPs would save a large amount of design time since we no longer need to deal with the internal structure construction and calculations.

3. Particles do not have rotation but in general rigid bodies do. In the current version of OSS we do not enable the HPs to have rotation because in our applications the HPs are the small fragments in a destructible environment and the process of destruction is a short animation from one equilibrium state to another.

4. The HPs have multiple linking nodes while a particle only has one node (itself) for connecting to. Therefore OSS offers more ways to construct the object based on our needs. This offers shape rigidity and stability which we have seen is lacking in PSSs. For example the cantilever can be made in an OSS as a sequence of connected block HPs whereas a sequence of spring connected massive particles will not stay horizontal.

The Assumptions

The purpose of using springs in the system is to allow deformations and fractures, and as a connection between HPs. The springs are just a way to apply forces between particles in the system and physical springs are not intended. This implies the following assumptions in OSS:

1. Springs in the OSS can pass through each other. This will prevent tangling when springs hit other objects or another spring.

2. Springs in the OSS model are always straight and therefore pass through Hard Parts to reach their connecting vertices.
The Data Structure
Every object in the simulation is made up of hard parts connected together by springs. In our method, we use a Standard Particle List (SPL) static array as the data structure to define and store all the vertices (also known as particles in physical meshes) of all the objects in the simulation. At each element of the SPL we store the position of the particle, its velocity, mass and the net force on it. Each hard part is a list of particles in the SPL. No two HPs can share the same particle from the SPL and every particle in the SPL must appear in exactly one HP list. All the soft parts are represented by a single list of springs. Each spring is represented by its two end particles in the SPL as well as its spring constant $k_s$ and its damping constant $k_d$. When a spring is broken it is deleted from the spring list. An example of the data structure is given below as shown in Figure 2.

![Figure 2: An Example of How the Connecting Springs and the Rigid Bodies are Defined By Referencing the Standard Particle List in Our Data Structure](image)

The Simulation Process
First we load the mesh data from an outside source file into the system. As described in 3.4 three data containers are constructed: a global standard particle list (static array), a hard parts list (rigid body list) and a spring list (for the soft parts).

Secondly we sum the applied net forces on each particle in each HP, and then calculate its displacement by using Newton’s law: $F = ma$.

Then we update the SPL to contain the new position and new velocity of every particle. (This maintains the HP constraints of constant separations and angles between particles in them.)

Thirdly we calculate the lengthening or shortening of every spring compared with its rest length, and then get the generated spring force on each single particle by using Hooke’s law for $F_s$ and add the damping force $F_d$ from the spring [Müller et al. 2008]. The net force on each particle in the SPL is the sum of the two forces $F_s$ and $F_d$ from every spring attached to the particle plus the force of gravity. After this we update the SPL to contain the new net force on each particle.

Steps two and three are repeated until the end of the simulation.

Fracturing
When an object is impacted by other objects or is affected by external forces, and the force generated by the impact or the force applied externally causes a spring to exceed its maximum length, an instant break of the spring will occur. Broken springs are deleted from the system. When all the springs between two parts of the object are broken, the object will be fractured and split into two or more separate fragments (which are the HPs that are now disconnected). Clearly HPs themselves do not fracture in our OSS model.

Integration Method and Stability
When dealing with the calculation of the HP’s
displacement, we use a fourth-order Runge-Kutta explicit integration method because each spring will only be shortened or lengthened by a relatively small distance (and then it will either break or the two ending particles will collide). Therefore there is no need to employ the implicit integration methods, which would have unconditional stability but would have more processing load.

OSS is a generalization of PSS in the sense that if all HPs in the OSS are single particles then it becomes the same as a PSS. We found from testing PSS structures with fixed nodes that they have the property of being totally elastic and formless. No arrangement of springs in the structure can retain a recognizable form and hold the model in static equilibrium while no other forces apply except gravity. Also a high damping coefficient is necessary in the PSS model to bring about static equilibrium without vibrations. On the other hand OSS models retain form until fracturing occurs and we find this more realistic for our applications. We have also mathematically proved that our hybrid system is a stable equilibrium system (see Appendix).

Collision Algorithm
Our research focus is on speed, and during the testing we found that because of the nature of our method (where in many cases we may deal with objects constructed from many HP units), the program will spend a large amount of time on self-collision detection and response processing and this slowed down the simulation significantly. Therefore we decided to employ a simplified collision detection and response method in the simulation to reduce the \( O(N^2) \) problem. To do this, we firstly set a ground level, and gave a “fixed” flag to each HP unit. Then we do the collision detection between each HP unit and the ground \( O(N) \). Once a HP unit hits the ground, its fixed flag is set to be true and no more displacement computations are done for it. Collision detection for axis parallel blocks is very fast so we set all HP units to be axis-parallel blocks. So secondly, we check the collision between HP units, the blocks. If one HP unit hits another HP unit on its top surface which is flagged by fixed = true, this HP will also be flagged by true and therefore it stops moving instantly. If one HP unit hits another HP unit which is flagged by fixed = true on a side surface, this HP will have its horizontal velocity components set to zero. With no horizontal velocity components, the HP will fall vertically to the ground or onto other fixed HPs. If two non-fixed HP blocks overlap no collision response processing is taken (and they can pass through each other).

Mesh Reconstruction After Fracturing
After fracturing, the original mesh is split into multiple parts and each new part is non-related and continues moving on its own. Normally the mesh has to be re-organized and new objects and their related representatives (rendering meshes, collision detection meshes and physical structure meshes) have to be created in the system. However in our system because we employed the SPL and referencing lists, there is no need to reconstruct the physical structure mesh. However the rendering mesh has to be re-constructed as it needs one or more new outer faces and textures to maintain a realistic rendering. However we are not concerned with this problem in this paper.

APPLICATION TEST EXAMPLES
In this paper we give two example cases of structures: a bridge and wall. Both objects are constructed by elementary rigid body (HP) units.
called “bricks” and springs. Figure 3A shows the physical mesh of a bridge before a high impact is applied at its centre and Figure 3B shows how the physical mesh of the bridge appears after it was broken by this impact. After the impact the structure had vibrations and deformations which quickly came to equilibrium as shown in Figure 3B. Figure 4A shows the physical mesh of a wall before being struck by a strong impulse equivalent to being hit in the front at the second top centre HP by a horizontal flying canon ball and Figure 4B shows the physical mesh of the wall after its destruction by this impact.

Our research only considers modeling the physical meshes (representing the OSS) and the collision detection and response method between the HPs and does not use the rendering meshes. Thus for this research we wanted to see the OSS behavior in the physical meshes only. In a real application there will be no obvious large gaps between rigid bodies (or display of springs) because we would use the rendering meshes surrounding the physical meshes for display and the physical meshes would be invisible.
CONCLUSION

In this paper we proposed a deformable object simulation method that generalizes a particle spring system by combining rigid body simulation and a breakable spring system. It offers a fast and visually realistic result to balance the performance and accuracy, and captures both deformation and fracture behavior. We used a global standard particle list and reference system to represent the object, and proposed a simplified collision detection and response method to reduce the calculations. The result achieves a smooth and acceptable realism in real-time rendering.

Future research can be done to look at the result of allowing rotations for HPs, looking at collisions between HPs and SPs, and between SPs and SPs, finding how to generate the physical meshes in multi-object simulations and their rendering and collision meshes, comparing OSS with FEM and to test the OSS on much larger and more complicated systems consisting of multiple objects.

ACKNOWLEDGMENT

This research is supported and funded by a La Trobe University Postgraduate Research Scholarship.

REFERENCES


This appendix reviews the mathematical analysis of the stability of a particle $P$ of weight $W$ located at position $(0,0)$ in 2D space supported by two springs $AP$ and $BP$ where $A$ is the point $(-b,-a)$ and $B$ is the point $(-b,a)$ (where $a$ and $b$ are positive real numbers) and gravity acts in the negative $y$ direction. From this setup the lengths of $AP$ and $BP$ are equal and denoted by $l$ where:

$$l = \sqrt{a^2 + b^2}$$

Both springs have the same spring constant $k$. Denoting the natural length of spring $AP$ by $L_A$ and the natural length of spring $BP$ by $L_B$ we have to find values for $L_A$ and $L_B$ such that the particle $P$ will be held in position (in equilibrium) at the origin. Then we have to determine whether this equilibrium is a stable or unstable equilibrium.

To determine the natural spring lengths for equilibrium we look at the forces on $P$:

$$F_x = kb\left(\frac{L_A + L_B}{l} - 2\right)$$

$$F_y = ka\frac{L_A - L_B}{l} - W$$

Therefore equilibrium is obtained for:

$$L_A = (1 + \frac{W}{2ka})l$$

$$L_B = (1 - \frac{W}{2ka})l$$

Next we have to consider small displacements $\delta x$ and $\delta y$ from equilibrium into disequilibrium in order to find the type of stability of this equilibrium point. Calculating the forces on $P$ at a general position $(x, y)$ gives:

---


APPENDIX (CONT.)

\[ F_x = k(x + b)\phi_A + \phi_B - 2 \]
\[ F_y = ky\phi_A + \phi_B - 2 + ka(\phi_A - \phi_B) - W \]
\[ \phi_A = -\frac{L_A}{\sqrt{(x+b)^2 + (y+a)^2}} \]
\[ \phi_B = -\frac{L_B}{\sqrt{(x+b)^2 + (y-a)^2}} \]

For \( x = y = 0 \) this reduces to the previous force equations. The force variation is therefore given by:

\[ \delta F_x = k(\phi_A + \phi_B - 2)\delta x + k(x+b)(\delta \phi_A + \delta \phi_B) \]
\[ \delta F_y = k(\phi_A + \phi_B - 2)\delta y + k(y+a)\delta \phi_A + k(y-a)\delta \phi_B \]
\[ \delta \phi_A = -\frac{L_A}{((x+b)^2 + (y+a)^2)^{3/2}} \{ (x+b)\delta x + (y+a)\delta y \} \]
\[ \delta \phi_B = -\frac{L_B}{((x+b)^2 + (y-a)^2)^{3/2}} \{ (x+b)\delta x + (y-a)\delta y \} \]

At equilibrium \( x = y = 0 \) we have \( 5\phi_A + 5\phi_B = 2 \) and these variations are:

\[ \delta F_x = kb(\delta \phi_A + \delta \phi_B) \]
\[ \delta F_y = ka(\delta \phi_A - \delta \phi_B) \]
\[ \delta \phi_A = -\frac{L_A}{I^3} (b\delta x + a\delta y) \]
\[ \delta \phi_B = -\frac{L_B}{I^3} (b\delta x - a\delta y) \]

Finally we find:

\[ \delta F_x = -2kb\frac{L_A}{I^3} \delta x \]
\[ \delta F_y = -2ka\frac{L_B}{I^3} \delta y \]

These equations say that for any small displacement (\( \delta x, \delta y \)) from the equilibrium position a small restorative force (\( \delta F_x, \delta F_y \)) acting in the opposite direction to the displacement and proportional to the displacement will be generated and therefore the particle at P supported by springs at A and B is in a stable equilibrium state.