Spatially coupled particle systems

SIGGRAPH'92 Course 16 notes: "Particle System Modeling, Animation, and Physically Based Techniques"

David Tonnesen

Department of Computer Science
University of Toronto, Canada M5S 1A4

Digital Equipment Corporation
Cambridge Research Lab, Cambridge, MA 02139

davet@dgp.toronto.edu

May 1992

1 Introduction

Particle systems, consisting of point masses moving under the influence of forces, model a variety of complex and time dependent phenomena. In computer graphics they have been used to model visually complex natural phenomena such as fire, foliage, waterfalls [20][27]. The complex behavior results from large numbers of independent particle reacting to forces such as gravity, vortex fields, and obstacles. In other approachs the particles interact through the use of spring and damping elements to produce deformable models which exhibit plasticity, bending, and fracture [30][10]. Recent models have borrowed ideas from molecular dynamics to model liquids and to mimic the effects of heat on solids [16][32][34]. By incorporating material dependent inter-particle constraints particle systems can be made to simulate specific properties such as those found in the draping of cloth [5].

We can categorize particle systems according to the interactions between particles. *Systems of independent particles* exist when the forces on each particle are independent of other particles in the system. More complex behavior results when the particles interact. Spring-mass systems are *particle systems with fixed particle coupling.* These systems are useful for modeling a variety of deformable materials but the ability to change the original geometry and topology is limited by the fixed set of connections between particles. In *particle systems with spatially defined particle coupling* the interactions between particles evolve dynamically over time. This results in the ability to model variable geometry and topology.

In these notes we concentrate on dynamically coupled particle systems. We show how these systems provide a new level of flexibility in interactive modeling, animation, and surface fitting problems.

2 What is a particle?

A simple particle consists of a *position* and a *mass*. By applying forces to each particle we can animate the particle system. Assuming a 2nd order dynamic system, each particle will exhibit *velocity*, *momentum*, and *acceleration*. We integrate the forces acting on a particle over time, in accordance with Newton's Laws of Motion, to create these dynamic attributes.

Other possible attributes include *color, size*, and *age*. Visually complex scenes can be created using a large number of particles and assigning each particle a color. The color attribute can be as simple as an color table index, a triplet of values to describe RGB color, or perhaps an index into a texture map. The concept of a particle with a *size* is intuitively interesting, but care must be taken when interpreting what this means. For dynamics it is simplest to consider a particle as a point mass and to incorporate distance constraints into potential energy functions [34]. Distance constraints may be also incorporated into the computation of surface geometry. When modeling complex natural phenomena, such as fire, a particle's *age* attribute has been used to control the creation of new particles and deletion of the current particle [20].

Orientations are used to arrange particles into surfaces instead of space filling volumes [29]. *Inertia*, *angular velocity*, and *angular acceleration* attributes result from incorporating orientations in a 2nd order dynamic system.

3 Particle dynamics

This section reviews the basic theory of dynamics as they apply to particle systems. We pay special attention to the case of oriented particles which require the use of multiple coordinate systems.

3.1 Position

A particle's position is naturally described by a vector. However a vector by itself does not describe a position, only a relative magnitude in a given direction. We need to use a frame of reference in combination with the vector in order to fully describe a position.

To define a frame of reference we arbitrarily choose a point in space we will call the origin and we choose a set of coordinate axes. By choosing the axes to be mutually orthogonal to each other we can describe any position in three space as a unique displacement from the origin. More precisely we choose a right handed orthonormal set of basis vectors. That is, each axis is represented by a basis vector of unit length which is perpendicular to the other two basis vectors and the vectors are labeled according to the right hand rule. It is best to use a common coordinate system to describe the position of all the particles and in accordance to common usage we will call this system "world coordinates".

3.2 Orientation

In computer graphics particles have been historically been defined without an orientation; only with position information. Points are infinitely small and thus it may be hard to imagine something infinitely small as facing a given direction. Then again these infinitely small points have mass, so then why not an orientation too?

Similar to how we created the world coordinate frame for the particle system we can create a local coordinate frame for each particle. We define the origin of the local coordinate frame to be the position of the given particle. We describe the orientation of the local frame with respect to the world coordinate frame: Neglecting the translation between the local and world frames, the

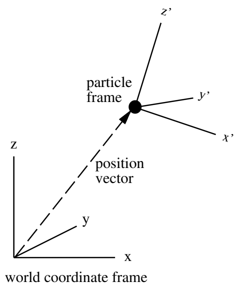

Figure 1: Particle in local and world coordinates.

orientation of the local frame is the rotation required to rotate the world frame such that the basis vectors of the two frames are aligned. Figure 1 illustrates the relationship between the local and the world coordinate frames.

We now look at three common ways of formally describing a rotation; First as a matrix, then as if a vector, and finally as a quaternion.

3.2.1 Rotation matrices

A rotation can be described by a 3x3 rotation matrix $\mathbf{R}$ where,

$$\mathbf{R} = \begin{pmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{pmatrix} \tag{1}$$

The first column of $\mathbf{R}$ describes the direction of the local coordinate frame's X axis in world coordinates. The second and third columns specify the directions of the local Y and Z coordinate axes respectively in world coordinates.

A vector is rotated using matrix vector multiplication. For example consider rotating the vector $\mathbf{x} = (0, 0, 1)$ in world coordinates by $\mathbf{R}$,

$$\mathbf{R}\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} r_{zx} \\ r_{zy} \\ r_{zz} \end{pmatrix}. \tag{2}$$

We find the Z axis in world coordinates is rotated to the direction of the Z axis in local coordinates, as expected.

Rotations specified by matrices are easily combined using matrix multiplication. A rotation of $\mathbf{R}_1$ followed by a rotation of $\mathbf{R}_2$ is equivalent to the matrix $\mathbf{R}_3$ where $\mathbf{R}_3 = \mathbf{R}_2\mathbf{R}_1$.

3.2.2 Rotation "vectors"

In three space a rotation is about an axis. Thus a natural way to describe a rotation is as a vector, where the direction of the vector describes the axis of rotation and the amount of rotation is encoded as the magnitude of the vector. Given the unit vector $\mathbf{a}$ along the axis of rotation and the angle of rotation θ we define a rotation $\mathbf{r}$ to be,

$$\mathbf{r} = \theta\mathbf{a} \tag{3}$$

Actually it is not correct to call the rotation a vector. Even though it has a magnitude and direction, thus looking like a vector, rotations do not follow the additive rule of vectors [13].

While this representation elegantly describes a rotation it is cumbersome to use it to represent the particle's orientation. First it is difficult to combine two successive rotations into a final single rotation. And second, to rotate a point by the rotation $\mathbf{r}$ one must without first convert $\mathbf{r}$ to another representation.

3.2.3 Quaternions

Quaternions are related to the rotation vector and solve the problem of combining successive rotations. We quickly present quaternions. For a detailed description we refer you to the papers by Shoemake [23][24].

Given the unit vector $\mathbf{a}$ along the axis of rotation and the angle of rotation θ, the unit quaternion that represents this rotation is given by the pair $\mathbf{q} = [s, \mathbf{w}]$ where s is a scalar and $\mathbf{w}$ is a vector,

$$s = \cos(\theta/2) \tag{4}$$

$$\mathbf{w} = \mathbf{a}\sin(\theta/2). \tag{5}$$

Successive rotations are easily represented using quaternion multiplication. If the quaternions $\mathbf{q}_1$ and $\mathbf{q}_2$ represent rotations then the product $\mathbf{q}_2\mathbf{q}_1$ represents the composite rotation of $\mathbf{q}_1$ followed by $\mathbf{q}_2$. Quaternion multiplication is given by the following rule,

$$[s_1, \mathbf{w}_1][s_2, \mathbf{w}_2] = [(s_1 s_2 - \mathbf{w}_1 \cdot \mathbf{w}_2), (s_1\mathbf{w}_2 + s_2\mathbf{w}_1 + \mathbf{w}_1 \times \mathbf{w}_2)]. \tag{6}$$

We rotate a vector using quaternion multiplication. We cast the vector (x, y, z) as a quaternion $\mathbf{p} = [0, (x, y, z)]$ where the s component is zero and the $\mathbf{w}$ component is the vector. To rotate the vector represented by $\mathbf{p}$, by the rotation represented by $\mathbf{q}$, we multiply $\mathbf{p}$ on the left with $\mathbf{q}$ and on the right with the inverse $\mathbf{q}^{-1}$,

$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1}. \tag{7}$$

The rotated vector (x', y', z') is contained in the new quaternion $\mathbf{p}' = [0, (x', y', z')]$. The inverse of a quaternion is given by

$$\mathbf{q}^{-1} = \frac{1}{(s^2 + \mathbf{w} \cdot \mathbf{w})}[s, -\mathbf{w}]. \tag{8}$$

3.2.4 Conversions

From the definitions given it is a simple matter to convert between quaternions and rotation "vectors". The mathematics behind converting between quaternions and rotation matrices is more complex. We refer you to [23] for an explanation and to [24] for **C** code. With these conversions in hand, the simplest way to convert between rotation "vectors" and rotation matrices is via quaternions.

3.3 Velocity

Since we are interested in animating the particle system we must consider how the position and orientation of the particles change over time. Thus we write the position and rotations as functions of time: $\mathbf{x}(t)$ and $\mathbf{r}(t)$.

3.3.1 Linear velocity

We define the linear velocity as the rate of change of the particle position over time. At time t the velocity of the particle is,

$$\mathbf{v}(t) = \frac{d}{dt}\mathbf{x}(t). \tag{9}$$

3.3.2 Angular velocity

We define the instantaneous angular velocity as the rate of change of the orientation over time,

$$\boldsymbol{\omega}(t) = \frac{d}{dt}\mathbf{r}(t). \tag{10}$$

In the form shown here the angular velocity $\boldsymbol{\omega}$ is a vector. The vector properties of addition and scalar multiplication hold for instantaneous angular velocities [13]. For example given two instantaneous angular velocities $\boldsymbol{\omega}_1$ and $\boldsymbol{\omega}_2$ the following holds,

$$\boldsymbol{\omega}_1 + \boldsymbol{\omega}_2 = \boldsymbol{\omega}_2 + \boldsymbol{\omega}_1. \tag{11}$$

Thus we can manipulate instantaneous angular velocities as vectors.

We could have derived angular velocity by differentiating the rotation matrix or quaternion but it is simpler to describe it in terms of the rotation $\mathbf{r}(t)$ as shown above. For matrix and quaternion derivations see [1]. Since we use angular velocity to approximate a rotation over a given time interval, the derivation shown above is ideal for our purposes. After computing the rotation we will convert it to another form.

3.4 Acceleration

3.4.1 Linear acceleration

The linear acceleration $\mathbf{a}$ is defined as the rate of change of the velocity

$$\mathbf{a}(t) = \frac{d}{dt}\mathbf{v}(t). \tag{12}$$

3.4.2 Angular acceleration

The angular acceleration is defined as the rate of change of the angular velocity,

$$\boldsymbol{\alpha}(t) = \frac{d}{dt}\boldsymbol{\omega}(t). \tag{13}$$

Similar to angular velocities, angular accelerations follow the rules of vector addition and scaling.

3.5 Force

The linear momentum $\mathbf{p}$ of a particle with mass m and velocity $\mathbf{v}$ is,

$$\mathbf{p}(t) = m\mathbf{v}(t) \tag{14}$$

The force on a particle is the change in momentum which is commonly written in terms of mass

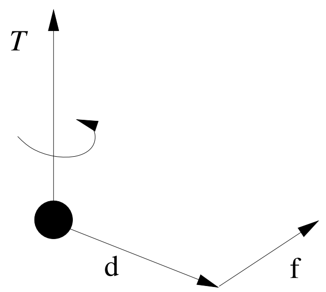

Figure 2: Torque results from force at a distance

and acceleration,

$$\mathbf{f}(t) = \frac{d}{dt}\mathbf{p}(t) = \frac{d}{dt}m\mathbf{v}(t) = m\frac{d}{dt}\mathbf{v}(t) = m\mathbf{a}(t). \tag{15}$$

3.6 Torque

The angular momentum of a particle with inertia $\mathbf{I}$ and angular velocity $\boldsymbol{\omega}$ is

$$\boldsymbol{\ell} = \mathbf{I}\boldsymbol{\omega}. \tag{16}$$

The inertia tensor $\mathbf{I}$, a 3x3 matrix, relates the angular momentum vector to the angular velocity vector by a linear transformation. In general $\mathbf{I}$ is a full 3x3 matrix, but we can simplify the system by setting the off diagonal elements to zero, in which case the inertia tensor represents a spherically symmetric particle.

The torque $\boldsymbol{\tau}$ on a particle is the rate of change in angular momentum and can be written in terms of inertia and angular acceleration,

$$\boldsymbol{\tau}(t) = \frac{d}{dt}\boldsymbol{\ell}(t) = \mathbf{I}\frac{d}{dt}\boldsymbol{\omega}(t) = \mathbf{I}\boldsymbol{\alpha}(t). \tag{17}$$

Torque results from force applied at a distance. The torque is defined to be the cross product of the distance and force vectors,

$$\boldsymbol{\tau} = \mathbf{d} \times \mathbf{f}, \tag{18}$$

as shown in Figure 2. Thus force applied to a particle results in both linear and angular accelerations.

4 Spatially coupled particle systems

4.1 Deformable models

Deformable solids have been modeled using hexahedral assemblies of point masses, springs, and damping elements [30]. In spring-mass models the springs are structural elements that hold the object together. As two particle connected by a spring are moved apart the force exerted by the spring steadily grows and pulls the particles back together. If we wish to model material that fractures we can set a distance threshold at which time the we say the spring breaks and there after we ignore the force generated by the spring. Springs are good for modeling solids that deform over small distances and are meant to maintain a given structure. However they are not the best representation for modeling materials in which the interactions between particles change over time. Using springs it is unclear how to model a material that we can arbitrarily split and join together into new and different configurations. For example, if we are going to model a fluid with point masses we expect the interactions between particles to be continually changing as the fluid flows.

Instead of a force that steadily grows as we pull two particles apart, we would like a force that encourages particles to maintain a given distance, and at the same time allowing groups of particles to be separated and joined back together at a later time in a new configuration. Ideally the potential energy function should act like a spring over small variations from the rest state and at farther distances allows particles to move freely past one another. In addition this function should be defined over *all particle pairs*, not a fixed set of pairs as in mass-spring systems. The result will be a modeling paradigm where we can easily modify a model's geometry without manually redefining the connections.

The Lennard-Jones potential energy function fulfills these criteria. It creates long-range attractive forces and short range repulsive forces which encourage particles to maintain an equal spacing and does not require the manual specification of inter-particle connections. The Lennard-Jones energy function defined as a function of separation r between a pair of particles,

$$\phi_{LJ}(r) = \frac{-\epsilon}{n-m}\left(m\left(\frac{r_o}{r}\right)^n - n\left(\frac{r_o}{r}\right)^m\right), \tag{19}$$

has a elliptical potential energy well local to r_o similar to the elliptical shape of the spring potential energy function.

When two particles are in equilibrium the potential energy between them is minimal. For the Lennard-Jones function this distance is defined as the equilibrium separation r_o. The energy required to separate two such particles is the *dissociation energy* $\phi(r_o) = \epsilon$. To increase the distance r between two particles by a small amount dr, the work that would have to be performed is $f(r)dr$ which can be equated to a loss in potential energy, that is,

$$f(r)dr = -d\phi(r). \tag{20}$$

For three dimensions we write the force as the variation in potential energy,

$$f = -\nabla E = -\left(\frac{\partial}{\partial x}E, \frac{\partial}{\partial y}E, \frac{\partial}{\partial z}E\right) \tag{21}$$

The total force F_i on a particle i is a result of summing all the interparticle forces f_{ij} between particles i and j,

$$F_i = \sum_i \sum_j f_{ij}(x_i - x_j). \tag{22}$$

The Lennard-Jones function is a spatially symmetric potential energy function. Given two particles p_i and p_j the set of minimum energy states (positions) for particle p_j relative to p_i is the locus of points in a sphere of radius r_o centered at the positions of particle p_i. For spherically symmetrical potential energy functions in 2-D, the particles arrange into hexagonal orderings. In 3-D the particles arrange into hexagonal ordered 2-D layers. Thus it is good for modeling volumes of material.

The Lennard-Jones energy function has been used to model both liquids [16][32] and solids [34]. For both material states attractive forces are necessary to bind the particles together. While it may not appear necessary to have attractive forces for fluids, it is the attractive forces that produce surface tension [35]. The repulsive forces are necessary to prevent particles from occupying the same space.

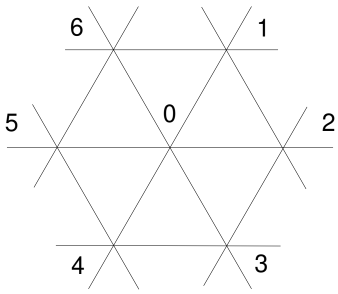

Figure 3: Finite difference grid for the Laplacian

By varying the dissociation energy ϵ we can model a continuous range of materials [34]. To create rigid solids we increase the dissociation energy. Decreasing the dissociation energy simulates materials which change shape easily. Further reduction will result in fluid like behavior. By varying the dissociation energy as a function of heat we create heat dependent models that can melt and freeze.

4.2 The heat equation for particle systems

The heat equation governs the flow of heat in a continuous material,

$$\frac{\partial \rho \sigma \theta}{\partial t} = \nabla \cdot (K \nabla \theta), \tag{23}$$

where $\rho(x, y, z)$ is the mass density of the body, $\sigma(x, y, z)$ is the specific heat, $\theta(x, y, z)$ is the temperature, and K is the thermal conductivity matrix. For a homogeneous and isotropic material $K = kI$, where I is the identity matrix the equation reduces to the familiar form,

$$\frac{\partial \rho \sigma \theta}{\partial t} = k \nabla^2 \theta, \tag{24}$$

where ∇^2 is the Laplacian.

From equation (23) we use a discrete approximation to solve for the heat and temperature of each particle over time. The heat of a particle is related to the temperature as follows. We assume the specific heat σ_i and temperature θ_i are constant over the interior of a given particle i. The mass m_i for the particle is equal to integrating the mass density ρ_i over the particle's volume. Heat ψ_i in particle i simplifies to, $\psi_i = \sigma_i \theta_i m_i$. The change in heat of particle i over a time interval approximates the left side of the heat equation.

The approximation for the right side of the heat equation is based on the finite difference method. To evaluate the $\nabla \cdot (K \nabla)$ term we introduce a thermal conductivity k_{ij} between each pair of particles i and j. For a 2-D hexagonal configuration of particles as labeled in Figure 3, the Laplacian of the temperature at the center particle $i = 0$ is

$$\nabla^2 \theta_0 \approx \frac{\frac{1}{6}(\theta_1 + \theta_2 + \theta_3 + \theta_4 + \theta_5 + \theta_6) - \theta_0}{\frac{1}{4} r^2} = \sum_{j=1}^{n=6} \frac{\frac{1}{6}(\theta_j - \theta_0)}{\frac{1}{4} r^2}, \tag{25}$$

where r as is the separation between particles [37]. For a 3-D configuration of hexagonal layers, the

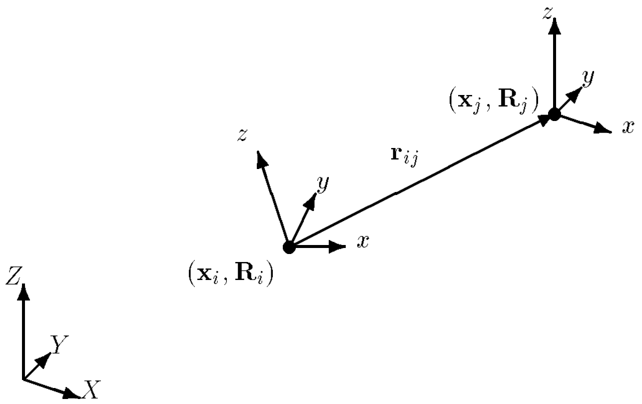

Figure 4: Global (X, Y, Z) and local coordinate frames (x, y, z). The global interparticle distance $\mathbf{r}_{ij}$ is computed from the global coordinates $\mathbf{x}_i$ and $\mathbf{x}_j$ of particles i and j. The local distance $\mathbf{d}_{ij}$ is computed from $\mathbf{r}_{ij}$ and the rotation matrix $\mathbf{R}_i$.

Laplacian of the temperature at the center particle $i = 0$ with surrounding particles $p_i : i = 1 \ldots 12$ is given by,

$$\nabla^2\theta_0 \approx \sum_{j=1}^{n=12} \frac{\frac{1}{12}(\theta_j - \theta_0)}{\frac{1}{4}r^2}. \tag{26}$$

4.3 Modeling Surfaces

Spatially symmetric potential energy functions encourage particles into tightly packed volumes making these good for modeling solids and fluids. Spatially assymetric potential energy functions can be used to encourage particles to arrange into sheets ideal for modeling surfaces.

To force particles to group into surface-like arrangements, we need new assymetric potential energy functions. Since each particle is intended to model a local surface area we associate with each particle an orientation. Each oriented particle defines both a normal vector (z in Figure 4) and a local tangent plane to the surface (defined by the local x and y vectors). More formally, we write the state of each particle as $(\mathbf{x}_i, \mathbf{R}_i)$, where $\mathbf{x}_i$ is the particle's position and $\mathbf{R}_i$ is a 3×3 rotation matrix which defines the orientation of its local coordinate frame (relative to the world coordinate frame). The third column of $\mathbf{R}_i$ is the local normal vector $\mathbf{n}_i$.

For surfaces whose minimum energy configurations are flat planes, we would expect neighboring particles to lie in each other's tangent planes. We can express this *co-planarity* condition as

$$\phi_P(\mathbf{n}_i, \mathbf{r}_{ij}) = (\mathbf{n}_i \cdot \mathbf{r}_{ij})^2 \beta(||\mathbf{r}_{ij}||). \tag{27}$$

where r_{ij} is the distance between particle i and j. The energy is proportional to the dot product between the surface normal and the vector to the neighboring particle (Figure 5a). We use a monotonically decreasing weighting function $\beta(r)$ to reduce the effects of distant neighbors. The co-planarity condition does not control the "twist" in the surface between two particles. To limit this, we introduce a *co-normality* potential

$$\phi_N(\mathbf{n}_i, \mathbf{n}_j, \mathbf{r}_{ij}) = ||\mathbf{n}_i - \mathbf{n}_j||^2 \beta(||\mathbf{r}_{ij}||), \tag{28}$$

which attempts to line up neighboring normals (Figure 5b). An alternative to surfaces which prefer

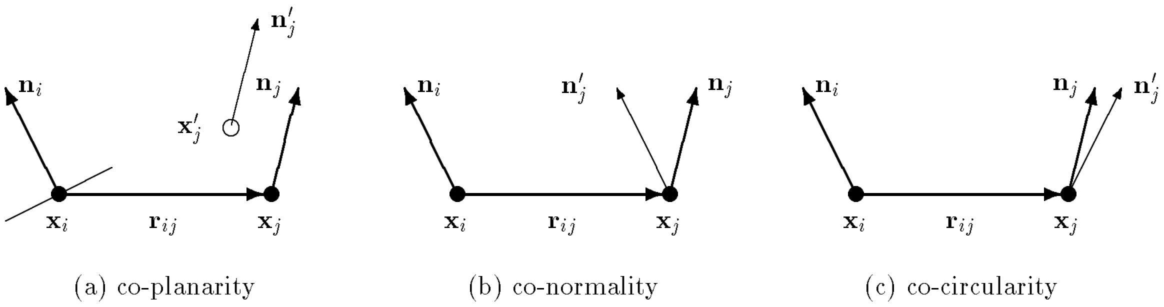

Figure 5: The three oriented particle interaction potentials. The open circles and thin arrows indicate a possible new position or orientation for the second particle which would lead to a null potential.

---

zero curvature or local planarity are surfaces which favor constant curvatures. This can be enforced with a *co-circularity* potential

$$\phi_C(\mathbf{n}_i, \mathbf{n}_j, \mathbf{r}_{ij}) = ((\mathbf{n}_i + \mathbf{n}_j) \cdot \mathbf{r}_{ij})^2 \beta(\|\mathbf{r}_{ij}\|) \tag{29}$$

which is zero when normals are antisymmetrical with respect to the vector joining two particles (Figure 5c). This is the natural configuration for surface normals on a sphere.

The above potentials can also be written in term of a particle's *local coordinates*, e.g., by replacing the interparticle distance $\mathbf{r}_{ij}$ by

$$\mathbf{d}_{ij} = \mathbf{R}_i^{-1}\mathbf{r}_{ij} = \mathbf{R}_i^{-1}(\mathbf{x}_j - \mathbf{x}_i), \tag{30}$$

which gives the coordinates of particle j in particle i's local coordinate frame. This not only simplifies certain potential equations such as (27), but also enables us to write use a weighting function $\beta(\mathbf{d}_{ij})$ which is not circularly symmetric, e.g., one which weights particles more if they are near a given particle's tangent plane. In practice, we have used

$$\beta(x, y, z) = c \exp\left(-\frac{x^2 + y^2}{2a^2} - \frac{z^2}{2b^2}\right) \tag{31}$$

with $b \leq a$.

To control the bending and stiffness characteristics of the deformable surface, we use a weighted sum of potential energies

$$E_{ij} = \alpha_{LJ}\phi_{LJ}(\|\mathbf{r}_{ij}\|) + \alpha_P\phi_P(\mathbf{n}_i, \mathbf{r}_{ij}) + \alpha_N\phi_N(\mathbf{n}_i, \mathbf{n}_j, \mathbf{r}_{ij}) + \alpha_C\phi_C(\mathbf{n}_i, \mathbf{n}_j, \mathbf{r}_{ij}). \tag{32}$$

The first term controls the average inter-particle spacing, the next two terms control the surface's resistance to bending, and the last controls the surfaces tendency towards uniform local curvature. The total internal energy of the system E_{int} is computed by summing the inter-particle energies

$$E_{\text{int}} = \sum_i \sum_j E_{ij}. \tag{33}$$

5 Computing Dynamics

To simulate the dynamics of the particle system we integrate the equations of motion through time. At each time step $t_{j+1} = t_j + \Delta t$ we sum the forces acting on each particle i and integrate over the time interval.

5.1 Controlling complexity

The straight forward evaluation of the inter-particle forces requires $O(N^2)$ computation. Highly accurate schemes which solve for all particle-particle interactions have been used by physicists and chemists [12, 8] and used directly they are usually too slow for animation or modeling applications. Since the far-field effects between particles are not critical for our purposes, we need only evaluate the interparticle forces between a particle and the nearest neighbors. A spatial subdivision approach can be used to subdivide space sufficiently so that we can find all of a particle's neighbors within some radius (e.g. $3r_o$ where r_o is the natural inter-particle spacing). To further reduce computation we can perform this operation occasionally and cache the list of neighbors for intermediate time steps.

5.2 Integration

The equations of motion result in a second order system of differential equations. We solve this as two coupled first order systems. Euler's method is a simple method to implement and understand[19]. Instead of using Euler's method directly on both systems we can make a better estimate for the second system by using the average of the old and new value (velocity) computed by the Euler step on the first system. The result is the Euler-Cromer method [7]. The resulting equations for linear and angular motion are shown below.

$$\begin{aligned}
\mathbf{a}(t) &= \mathbf{f}(t)/m && (34)\\
\mathbf{v}(t+\Delta t) &= \mathbf{v}(t) + (\Delta t)\mathbf{a}(t) && (35)\\
\mathbf{x}(t+\Delta t) &= \mathbf{x}(t) + (\Delta t)\frac{\mathbf{v}(t+\Delta t)+\mathbf{v}(t)}{2} && (36)\\
\boldsymbol{\alpha}(t) &= \mathbf{I}^{-1}\boldsymbol{\tau}(t) && (37)\\
\boldsymbol{\omega}(t+\Delta t) &= \boldsymbol{\omega}(t) + (\Delta t)\boldsymbol{\alpha}(t) && (38)\\
\mathbf{r} &= (\Delta t)\frac{\boldsymbol{\omega}(t+\Delta t)+\boldsymbol{\omega}(t)}{2} && (39)\\
\mathbf{q}_r &\Leftarrow \mathbf{r} && (40)\\
\mathbf{q}(t+\Delta t) &= \mathbf{q}(t)\mathbf{q}_r && (41)
\end{aligned}$$

The equations for angular motion are not as straight forward as the equations for linear motion. The change in orientation is computed as a rotation "vector" $\mathbf{r}$ from the angular velocity $\boldsymbol{\omega}$. The rotation $\mathbf{r}$ then is converted into the quaternion $\mathbf{q}_r$ before modifying the particles current orientation.

More sophisticated numerical integration techniques such as Runge-Kutta [19] or semi-implicit methods [31] could also be used, and would result in better convergence and larger time steps, but at the expense of a more complicated implementation.

6 Rendering

The computation of a continuous surface description over any local area requires information from several particles and thus will be slower than rendering each particle separately. Rendering, as we know from the ray-tracing and radiosity literature, can be very time consuming. However for modeling and animation we desire an interactive system. We look at rendering from two approachs: First to create real-time informative displays of the system state and second to create renderings of high quality continuous surfaces.

6.1 Visual cues for real-time use

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

6.1.1 Light emitting points

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

6.1.2 Wire frame particles

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

6.1.3 Surface approximations

To quickly approximate the surface of a solid model the particles can be displayed as spheres or cubes. To approximate the surface resulting from oriented particles the particles can be displayed as filled hexagons. Displaying the neighbor connections between particles as line segments is another method for quickly approximating the surface while at the same time conveying the structure of the particle system. A slightly longer process is to display a wire triangulation of the surface (see section 6.2.2).

6.2 Constructing continuous surface descriptions

Dynamically coupled particle systems do not inherently possess a surface description. Instead a surface description must be computed from the state of the particle system and a suitable algorithm.

6.2.1 Surfaces for volume models

Implicit surfaces

We use implicit surfaces for our particle based model of solids and fluids. An implicit surface is the set of points in space that satisfy an equation of the form, $f(x, y, z) = 0$, where f is a function defined in 3-space.

To define an implicit surface for the particle system we associate a field function with each particle. Each function is based on distance from the particle. We define the surface of the particle system to be all points is space where the summation of the individual field functions equals a given threshold, that is we define G as the implicit surface equation for the particle system,

$$G(x, y, z) = \sum_i g_i(x, y, z) - threshold = 0 \tag{42}$$

where g_i is the field function for particle i. Blinn [2] introduced this concept (defining surfaces based on points and associated field functions) to computer graphics and hence this model is commonly referred to as "Blinn blobbies" or simply "blobbies". Blinn used an exponential field function to create smoothly varying surfaces. Polynomial functions similar in shape to Blinn's function have also been used with good results [33][38].

Direct rendering

In order to make a direct rendering of the iso-surface we need to be able to find the intersection of a ray with the surface. We compute the intersection points of a ray with the surface, by combining the ray equation with the implicit surface equation (42) and solving for the roots. The problem is complicated by the fact that for a system of N particles there may be as many as $2N$ intersections with a given ray. For the rendering of opaque surfaces, this problem can be avoided by finding the first intersection of the ray with the surface. Assuming each particle field function is monotonically decreasing we can guess an interval along the ray where the first intersection will occur [2]. Then we find the intersection using an iterative root solver. If we restrict the field functions to be monotonically decreasing polynomials with a limited range, then we can find all of the intersections of a ray with the surface [33][39]. The approach is to split the ray into non-overlapping intervals such that roots over each interval can be analytically solved for. Since we analytically solve for the roots the method is considerably faster than an iterative approach. It is also suitable for CSG (constructive solid geometry) modeling systems. The use of CSG requires that all the intersection points of a ray and the surface be returned. The arches shown in Figure 7 are rendered directly as an iso-surface.

Polygonal approximations

We can also render a polygonal approximation of the iso-surface. The polygonization process starts by sampling points in space and categorizing the points as inside or outside of the surface. Surface intersection points are interpolated along the line segment connecting two samples of opposite in/out value. Well known polygonization techniques [38][15] sample over a cubic grid of points. Unfortunately, ambiguous polygonizations occur since more than one possible plane will match certain combinations of in/out vertices [6]. These cases can be resolved by testing additional points[38]. By sampling and testing the vertices of a tetrahedron (a 3D simplex) instead of a cube the ambiguous polygonization problem disappears [3][36]. Adaptive polygonizations benefit from concentrating polygons in areas of high curvature while limiting the number of polygons in areas of low curvature [3][36][9].

6.2.2 Surfaces for oriented particles

We render an oriented particle system by generating a triangulation over the set of points. For shaded rendering, Gouraud, Phong, or flat shading can be applied to each triangle. For a smoother looking surface, a cubic patch can be created at each triangle (since we know the normals at each corner).

Intuitively a triangulation should avoid long thin triangles in favor of triangles that are "more or less equilateral". The *Delaunay triangulation* accomplishes this by maximizing the minimum interior angle over all possible triangulations [14]. In 2D it is defined as the unique triangulation such that the circumcircle[1] of each triangle does not contain any other point in its interior. The Delaunay triangulation can be computed directly using a recursive procedure or as the dual of the Voronoi diagram. Both approachs take $O(N \log N)$ optimal time [18].

The three dimensional Delaunay triangulation is an $O(N^2)$ collection of tetrahedra tessellating the interior of the convex hull [4]. The convex hull is the smallest polyhedra such that all the points are contained with in the volume. The 2D circumscribing circle rule naturally extends to a 3D rule based on the circumscribing sphere. Since we do not want to generate all the triangles in the 3D triangulation we use a simplified version of the Delaunay triangulation. We find all triangles up to a maximum size (edge length) where no other points are inside of the circumscribing sphere. This results in surfaces that breaks as the points are moved far enough apart.

7 Applications

7.1 Geometric modeling

For shape design and rapid prototyping applications we require a highly interactive system which does not force the designer to think about the underlying representation or be limited by its choice. For example, we require the basic abilities to join several surfaces together, to split along arbitrary boundaries, or to extend existing surfaces together without specifying exact connectivity. Spatially coupled particles provide such features.

There are numerous modeling paradigms that can be employed. We can "mill" a solid block of material into a given shape by deleting all particles which lie outside of a given implicit surface definition. From a geometric surface description we can "cast" solids by pouring liquid particles into a mold and then cool the particles into a solid. A user can "sculpt" a model by interactively adding, deleting, and deforming the model. Local sections of the model can be heated to soften the material similar to how an artist creates a sculpture in wax before casting it in metal.

The gable model shown in Figure 7 was interactively built by creating particles and joining them with other particles. Figure 6 shows how separate sheets of particles are joined together under user control with a moving tool. We use a variety of geometric shapes as tools which can move, erase, attract or repel particles. With a stretching rule we create elastic sheets which can be infinitely stretched and deformed into a desired shape [29]. Figure 8(b) shows how a sheet of particles has started to stretch.

[1] The circumcircle of a triangle is the circle such that each vertex is on the perimeter of the circle. There is exactly one circumscribing circle for a given triangle.

7.2 Animation

Spatially coupled particle systems are useful in animation when incorporated into an environment with other objects and external forces. Collisions with surfaces are easily implemented using reaction-constraint forces [17], repulsion forces [34], or the penalty method [31].

A variety of materials can be simulated by varying the properties of the potential energy functions. In Figure 7 the material characteristics are varied by changing the "width" of the Lennard-Jones potential energy well. In Figure 8 a sheet of particle is shown to act as a draping cloth, a sheet of elastic material that is stretching, and as a sheet of material that tears. The addition of heat further extends the range of materials that are modeled.

7.3 Surface fitting

We can use oriented particles to fit surfaces to sparsely sampled 3D point data. In Figure 9 a toroidal surface is interpolated through a set of seed points. The resulting particle surface can be triangulated to generate a continuous surface description. The toroid was sparsely sampled using only 300 points which is significantly less dense than the sampling in most remotely sensed data (such as range images, height fields, and CAT scan volume data sets). Oriented particles have the additional benefit in that they are not restricted by surface topology.

References

[1] D. Baraff. Rigid body concepts. In *SIGGRAPH '91 Course notes #23: An Introduction to Physically Based Modeling*. SIGGRAPH, July 1991.

[2] J. F. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.

[3] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5(4):341–355, November 1988.

[4] J.-D. Boissonnat. Representing 2D and 3D shapes with the Delaunay triangulation. In *Seventh International Conference on Pattern Recognition (ICPR'84)*, pages 745–748, Montreal, Canada, July 1984.

[5] D. Breen, D. House, and P. Getto. A particle-based computational model. In *Scientific Visualization of Physical Phenomena (Proceedings of Computer Graphics International '91)*, pages 113–134. Springer-Verlag, June 1991.

[6] M. Dueurst. Additional reference to 'marching cubes'. *Computer Graphics*, 22(2):72–73, April 1988.

[7] H. Gould and J. Tobochnik. *An Introduction to Computer Simulation Methods*. Addison-Wesley, Reading Massachusetts, 1988.

[8] L. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems*. The MIT Press, Boston, Massachusetts, 1988.

[9] M. Hall and J. Warren. Adaptive polygonalization of implicitly defined surfaces. *IEEE Computer Graphics and Applications*, 10(6):33–42, November 1990.

[10] D. Haumann, J. Wejchert, K. Arya, B. Bacon, A. Khorasani, A. Norton, and P. Sweeney. An application of motion design and control for physically-based animation. In *Proceedings of Graphics Interface '91*, pages 279–286, June 1991.

[11] J. S. Hersh. Soft object extensions to the clockworks system. Technical Report TR-87054, Rensselaer Design Research Center, Rensselaer Polytechnic Institute, Troy, New York, 1987.

[12] R. W. Hockney and J. W. Eastwood. *Computer Simulation using Particles*. McGraw-Hill Inc., New York, student edition, 1988.

[13] B. Hoffmann. *About Vectors*. Dover Publications, New York, 1966.

[14] C. L. Lawson. Software for C^1 surface interpolation. In *Mathematical Software III (Proceedings of the Symposium on Mathematical Software '77, Madison, Wisconsin)*, pages 161–194, March 1977.

[15] W. Lorensen and H. Cline. Marching cubes: A high resolution 3-D surface construction algorithm. *Computer Graphics (SIGGRAPH'87)*, 21(4):163–169, July 1987.

[16] G. Miller and A. Pearce. Globular dynamics: A connected particle system for animating viscous fluids. In *SIGGRAPH '89, Course 30 notes: Topics in Physically-based Modeling*, pages R1 – R23. SIGGRAPH, August 1989.

[17] J. C. Platt and A. H. Barr. Constraint methods for flexible models. *Computer Graphics (SIGGRAPH '88)*, 22(4):279–288, August 1988.

[18] F. P. Preparata and M. Shamos. *Computational Geometry - An Introduction*. Springer-Verlag, New York, NY, 1985.

[19] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, England, 1988.

[20] W. T. Reeves. Particle systems—a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics*, 2(2):91–108, April 1983.

[21] W. T. Reeves. Particle systems a technique for modeling a class of fuzzy objects. *Computer Graphics*, 17(3):359–376, July 1983.

[22] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, Massachusetts, 1989.

[23] K. Shoemake. Animating rotation with quaternion curves. *Computer Graphics (SIGGRAPH'85)*, 19(3):245–254, July 1985.

[24] K. Shoemake. Quaternion calculus for animation. In *SIGGRAPH '89, Course notes #23: Math for SIGGRAPH*, pages 187 – 205. SIGGRAPH, August 1989.

[25] K. Sims. Particle dreams. In *SIGGRAPH Video Review*, New York, August 1988. ACM SIGGRAPH.

[26] K. Sims. Particle animation and rendering using data parallel computation. In *SIGGRAPH '89, Course 30 notes: Topics in Physically-based Modeling*, pages T1– R15, New York, August 1989. SIGGRAPH.

[27] K. Sims. Particle animation and rendering using data parallel computation. *Computer Graphics (SIGGRAPH'90)*, 24(4):405–413, August 1990.

[28] R. Szeliski and D. Tonnesen. Surface modeling with oriented particles. Technical Report TR 91/14, Digital Equipment Corporation, December 1991.

[29] R. Szeliski and D. Tonnesen. Surface modeling with oriented particles. To appear in *Computer Graphics (SIGGRAPH'92)*, 26, July 1992.

[30] D. Terzopoulos and K. Fleischer. Deformable models. *The Visual Computer*, 4(6):306–331, December 1988.

[31] D. Terzopoulos, J. Platt, A. Barr, and K Fleischer. Elastically deformable models. *Computer Graphics (SIGGRAPH'87)*, 21(4):205–214, July 1987.

[32] D. Terzopoulos, J. Platt, and K. Fleischer. Heating and melting deformable models (from goop to glop). In *Proceedings Graphics Interface*, pages 219–226. Graphics Interface, June 1989. (reprinted in Journal Of Visualization And Computer Animation 1991).

[33] D. Tonnesen. Ray-tracing implicit surfaces resulting from the summation of bounded polynomial functions. Technical Report TR-89003, Rensselaer Design Research Center, Rensselaer Polytechnic Institute, Troy, New York, January 28, 1989.

[34] D. Tonnesen. Modeling liquids and solids using thermal particles. In *Proceedings Graphics Interface '91*, pages 255–262, June 1991.

[35] D. H. Trevena. *The liquid phase.* Wykeham Publications, London, England, 1975.

[36] L. Velho. Adaptive polygonization of implicit surfaces using simplical decomposition and boundary constraints. In *Proceedings of Eurographics '90*, pages 125–136, 1990.

[37] E. Vitasek. *Survey of Applicable Mathematics*, chapter 27: Solution of Partial Differential Equations by the Finite-Difference Method. MIT Press, 1969.

[38] B. Wyvill, C. McPheeters, and G. Wyvill. Data structures for soft objects. *The Visual Computer*, 2:227–234, 1986.

[39] G. Wyvill and A. Trotman. Ray-tracing soft objects. *Proceedings of Graphics Interface 1989*, pages 469–476, 1989.

A Programming hints

This appendix provides hints of where to start if you were going to code up a spatially coupled particle system. *Disclaimer: The* **C** *data structures and code are for illustration purposes and has not been tested.*

A.1 Data structures

There are several ways we could arrange the data describing a particle. A good approach is to group the attributes for a particle together into a single structure. A possible structure in **C** for an *oriented particle* is shown below. Note we have defined data types for *vectors*, *matrices*, and *quaternions*.

```
typedef float[3] vec;
typedef float[3][3] matrix;
typedef float[4] quat;

struct particle
{
   vec    rgb;      /* RGB color */
   float  mass;     /* mass */
   vec    f;        /* force accumulator */
   vec    v;        /* velocity at time t */
   vec    p;        /* position at time t */
   matrix I;        /* inertia matrix */
   vec    t;        /* torque accumulator */
   vec    w;        /* angular velocity at time t */
   quat   q;        /* orientation at time t */
};
```

We can store the system of particles as an array of `particle` structures with an integer holding the current number of particles. Recomputing the forces between particles is simplified by storing the current connections between particles as a list of edges. For each pair of particles that are "close" we store the indices of the two particles as an edge in the edge list.

```
#define MAX_P  1000
#define MAX_E  MAX_P * 6

typedef int[2] edge;

particle particles[MAX_P];   /* array of particles */
int num_p = 0;               /* current number of particles */

edge edges[MAX_E];           /* edges (pair of interacting particles) */
int num_e = 0;               /* current number of edges */
```

A.2 Psuedocode for main loop

1. Initialize parameters such as time, Δt, viewing parameters, graphics window, etc...
2. Create particles
3. *loop* {
   (a) Compute neighbors and save in edge list.
   (b) Traverse edge list computing inter-particle forces and torques.
   (c) Compute external forces.
   (d) Update particle's state (position, velocity, ...) by integrating over Δt.
   (e) Process user input.
   (f) Draw scene.

   }

A.3 Naive neighbor computation

A simple neighborhood computation scheme is shown below. Given N particles it checks the distance between each pair of particles to see if they are within the specified distance r. If particle separation is less than r then the particles are said to be neighbors and an edge describing this pair is saved in the edge list. While this routine will do for small systems of particles, in general more efficient techniques should be used. Samet [22] provides a thorough discussion of spatial data structures and algorithms.

```
/*
    This is a simple N^2 algorithm for finding all pairs
    of particles that are separated by ``r'' distance or less.
*/
int find_neighbors (particle *p, int n, float r, edge *e)
{
   float r_sq, d_s;
   int num_e = 0;
   vec d;
```

```
   r_sq = r * r;    // Use distance squared (optimization to avoid sqrt).

   for (int i = 0; i < n; i++)
      for (int j = 0; j < i; j++)
      {
         vec_sub (p[i].p, p[j].p, d);
         d_sq = vec_magn_sq(d);
         if (d_sq <= r_sq)
         {
            edges[num_e][0] = i;
            edges[num_e][1] = j;
            num_e++;
         }
      }
   return num_e;
}

float vec_magn_sq (vec *V)
{
   return (V[0]*V[0] + V[1]*V[1] + V[2]*V[2]);
}

void vec_sub (vec a, vec b, vec c)
{
   c[0] = a[0] - b[0];
   c[1] = a[1] - b[1];
   c[2] = a[2] - b[2];
}
```

A.4 Updating particle state

After we have computed the forces acting on a particle at time t we can approximate the particle's new state at time $t+\Delta t$ by integrating over the time step Δt. See section 5.2 for formulas to compute a particle's new state (position, velocity, orientation, and angular velocity) from a force and torque.