Obtaining Fuzzy Representations of 3D Objects

Technical Report by
Brent M. Dingle

Department of Computer Science
Texas A&M University
November 2005

Abstract. This paper presents a method to convert 3D objects into a dense particle representation. To accomplish this we assume a pre-existing 3D object is supplied. Through the use of forces based on Lennard-Jones potentials and Newtonian physics we fill the object with particles of a given size. The result of this filling offers a well defined 3D particle representation of the object which may be used in a variety of ways. Such a representation should allow more effects to be performed on the object than otherwise would be available. This method should easily fit into modeling systems based on particle systems. It may also be extended to make use of the GPU of a graphics card or ported to a parallel processing environment to increase performance.

1 Introduction

The intent of this paper is to present a method which allows a given 3D object to be converted into a particle representation. This representation will allow various effects to be performed on the object which would not normally (or easily) be possible in any other form, as will be discussed in the next section. The method we use to convert the given object is a filling effect driven by forces based on Lennard-Jones (LJ) potentials and Newtonian physics. The LJ potentials are used to assist in controlling the denseness of the filling. This is a specific implementation of a general algorithm, both of which will be presented.

The advantages of this implementation are:

- It easily fits into any preexisting particle-based modeling system.
- It is physically based.
- It offers a limited controllability of the denseness of the packing.
- It works with most well defined 3D objects, regardless of concavity or holes.
- It can be extended to use the GPU of the computer's graphics card to increase performance.
- It can be ported to a parallel processing environment for greater performance gains.

For this paper we begin with a brief discussion of the previous work, background and the direction we will be heading with this method. In section 3 we present a general algorithm used to fill an object with particles. In sections 4, 5 and 6 we present the details of our specific implementation of the general algorithm. We conclude in section 7 with a brief summary and possible future usage of this method.

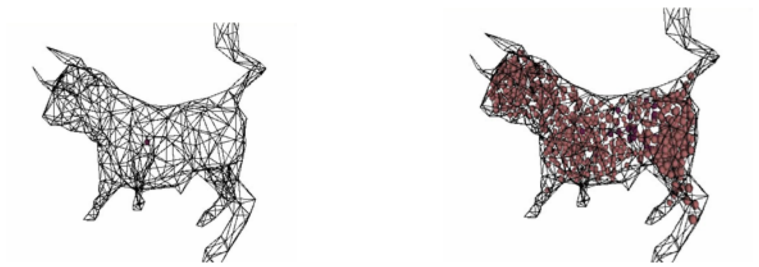

Fig. 1. A bull in the process of being filled with particles.

2 Background and Direction

Particle systems are used for a variety of objectives. In computer graphics they often represent fuzzy or poorly defined objects. For example, many people have used particle systems to model gases and liquids [Reev1983, Eber1994, Stam1999, Yngv2000, Fedk2001]. They have also been used to model various types of cloths [Bree1994, Bara1998, Choi2002] and other deformable objects [Chri1997, Terz1988, Barr2000]. There has also been an interest in using them to allow molding and sculpting of objects [Coqu1990, Terz1994]. Oddly there has been very little discussion on how to obtain particle representations of already existing objects. It is noted there is an effort to tetrahedralize objects to be used as deformable objects [Meul2002]. However, not much beyond that type of a explicit volumetric representation has been discussed, whereas there has been a significant discussion on surface representations [Szel1992, Pfis2000].

In the past, in most applications, such as those systems modeling gases, liquids or other amorphous substances, representing a specific object shape would not be a major concern. In other cases such as with cloth, the initial starting form of the shape is often easy to explicitly define. However in the case of deformable objects, and to a limited degree molding and sculpting scenarios, being able to apply particle type effects to complicated, already constructed, objects is desirable.

So with the growing interest in modeling deformable objects, and an ever present interest in special effects, an investigation of representations useful to such endeavors is worth undertaking. Specifically, being able to convert a given object into a particle representation would be extremely useful as it would allow effects to be performed on the object that would not normally (or easily) be allowed. For example, to melt, bend, break, disintegrate, or turn an object into smoke might be easier if the object were represented as a system of particles. Of course it can be argued that other representations allow other effects to be done in a more efficient manner than a particle representation. While we cannot deny this, we simply make the point that particle representations have advantages too. We also note that operations on systems of particles may have straightforward implementations in parallel environments [Sims1990] and can be streamed to the GPU of a computer's graphics card to improve performance of such operations.

In the filling method we present shortly, we offer a limited controllability of the denseness of the packing. This control is in two forms. First the size of the particles themselves will effect how dense and how well an object is filled. The second control is in the "settings" for the LJ potentials controlling the interparticle reaction. This control may be desirable for achieving different effects or assisting in performance control.

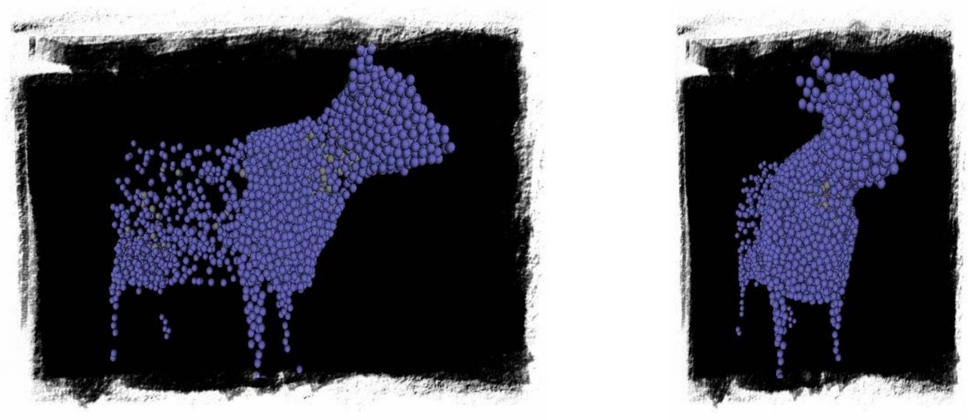

Fig. 2. A cow in the process of being filled with particles.

3 The General Method

To convert a 3D object into a particle representation we assume we have been given a well defined object. By well defined it must be possible to determine if a point is inside, outside or on the surface of the object. The surface of the object must not have any holes. If the object itself has holes, such as a torus, the surface of the object must accurately reflect this. An example of a well defined object would be a vertex-face list of triangles that represent the surface of the object.

We will convert the object into a particle representation by filling the object with particles of a given radius. This filling process is to be a physically based process. It may be governed by a variety of forces. Our specific implementation will use forces based on Lennard-Jones potentials and Newtonian physics, as described later in this paper. The general filling algorithm we will be using is:

```
While (stopping criteria not true)
   Add a particle to the system.
   Update the System.
End While
Set MaxParts = Current Number of Particles minus 1.
While (number of particles < MaxParts)
   Add a particle to the system.
   Update the System.
End While
While (system not at rest)
   Update the system.
End While
```

This is a two pass process. There are several different ways to define the “stopping criteria.” The most obvious choice in this situation is to use an “overfull” test of some kind. We will describe two such tests below. In the final while-loop, the system should be considered at rest when the particles stop moving. The process of updating the system will depend on what forces are driving the system. In most implementations this process will involve updating the state of each particle across a finite time step. Thus an Euler, Midpoint, Runge Kutta or similar integration method will be part of the update process. Likewise, as the particles should be moving, some form of collision detection and response will also be needed in the update process.

4 Stopping Criteria

In our specific implementation we discovered two stopping criteria that worked well. The first criterion is based on the total kinetic energy, or the total sum of motion of particles. It proves useful for a system containing a large number of particles. In general it works because of the way our collision detection is implemented: if all the particles cannot be moved into a non-collision state then each particle is moved into a new location. This causes a detectable increase in the motion of the system. When this happens across several consecutive time steps, the object may be overfull. So we remove one particle and wait several time steps for the system to become more stable. Once the system regains stability we attempt to add the removed particle back into the system. This process is repeated several times. If the system repeatedly loses stability a given number of times then the object is considered overfull and the “stopping criteria” for the first while loop is met.

The second criterion we found useful works best for systems with a small number of particles. This criterion is based on the assumption if there are a small number of particles in a large area of free space then there should be no reason for the particles to stay in contact for a prolonged amount of time. The test itself is just a matter of counting the number of collisions between particles. If the number of collisions stays constant for a given number of successive iterations (a given time period) then it is likely the particles cannot move out of collision. So as with the first criterion, we implemented a process of removing a particle, waiting several iterations and trying again. If the collision count cannot be reduced then we assume the object is overfull and the “stopping criteria” for the first while loop is met.

5 The Forces

In our implementation each particle has a position, velocity, acceleration, radius and mass. Each particle is introduced to the environment one at a time. Each is “born” at the same location within the object. Each begins with a random velocity within a given range. The particles, thus, may collide with each other and the boundary represented by the surface of the object to be filled. We use inelastic collisions, thus the velocity of each particle is reduced upon each collision. In all operations we guarantee that momentum is conserved. As there exist no constant acceleration forces, such as gravity, within the system, the velocity of each particle will tend to zero with time. The only way the momentum of the system may increase is when a new particle is introduced to the system. We also apply interparticle forces based on potentials similar in form to Lennard-Jones (LJ) potentials.

A detailed description of LJ potentials can be found in [Rapa2004]. The important quality we need them for is the bonding they create between the particles. This bonding helps promote a dense packing of the particles within the object. By altering the exact nature of the potential we may loosely control how densely the particles fill the object. In general we will use extremely strong repulsive forces to prevent particles from overlapping, and significantly weaker forces to attract them together. The general graph of the potential energy between two particles is given by:

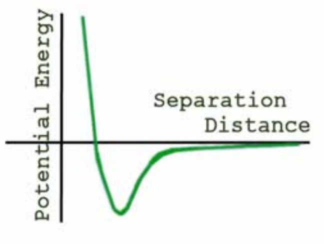

Fig. 3. Potential energy between two particles.

The generic form of the equation we use to represent the potential between two particles is:

$$u(r_{ij}) = \begin{cases} 4\varepsilon\left(\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right) & r_{ij} < r_c \\ 0 & r_{ij} \geq r_c \end{cases} \qquad \textbf{(1)}$$

Where $\mathbf{r}_i$ is the location in space of particle i and $r_{ij} = |\, \mathbf{r}_i - \mathbf{r}_j \,|$. In this relationship the particles will repel each other when close together, attract each other when slightly apart and eventually have no effect on one another at a distance of r_c. In this relation, the value of ε will control the strength of the relationship between the particles and σ will define a length scale [Rapa2004]. The exponents on this equation may be changed to produce a variety of effects between the particles. In general r_c will be chosen so that $u(r_c) = 0$. This can be achieved by setting $r_c = (\sigma * 2^{1/6})$ and ignoring the attractive tail to obtain:

$$u(r_{ij}) = \begin{cases} 4\varepsilon\left(\left(\frac{\sigma}{r_{ij}}\right)^{12} - \left(\frac{\sigma}{r_{ij}}\right)^{6}\right) + \varepsilon & r_{ij} < 2^{1/6}\sigma \\ 0 & r_{ij} \geq 2^{1/6}\sigma \end{cases} \qquad \textbf{(2)}$$

So when $r_{ij} < 2^{1/6}\sigma$ the vector force particle j exerts on particle i is:

$$\mathbf{f}_{ij} = -\nabla u(r_{ij}) = \left(\frac{48\varepsilon}{\sigma^2}\right)\left[\left(\frac{\sigma}{r_{ij}}\right)^{14} - \left(\frac{1}{2}\right)\left(\frac{\sigma}{r_{ij}}\right)^{8}\right]\mathbf{r}_{ij} \tag{3}$$

And when $r_{ij} \geq 2^{1/6}\sigma$ the force is zero.

This allows us to calculate an acceleration for each particle using F = $m\mathbf{a}$, specifically:

$$m_i\mathbf{a}_i = \mathbf{f}_i = \sum_{\substack{j=1 \\ j\neq i}}^{n} \mathbf{f}_{ij} \tag{4}$$

Where m_i is the mass of particle *i*,, $\mathbf{a}_i$ is the acceleration of particle *i*, and $\mathbf{f}_i$ is the force on particle *i*.

We have chosen to use forces based on LJ potentials for the control they afford. Specifically, by changing the constants of these equations a limited control of the density of the packing is achieved. Obviously changing the radius of the particles will also offer a limited control. Together these allow for some desirable results. For example if there are narrow passages in the object small particles are needed, however if the object is large in volume just using small particles results in a large number of particles, which will slow the process down. By using forces based on LJ potentials we have the option of reducing the particle size and increasing the rest distance between the particles. This allows the particles to traverse the narrow passages and keeps the quantity of particles lower. So careful balancing of these two controls results in a nice packing using a reasonable number of particles.

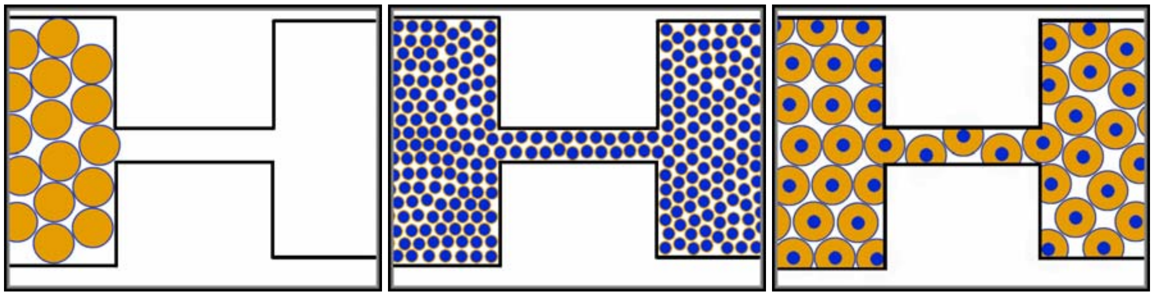

Fig. 4. Narrow passage filling with large particles, small particles and small particles with large LJ resting distance.

6 Updating

The final component of our implementation is the update process. This can be outlined as follows:

```
Store the old state of each particle.
Set other variables as needed.
Perform pairwise check between particles.
     Apply attractive, repulsive and collision forces as needed.
For each particle check for boundary collisions.
     Apply collision forces/response as needed.

Perform integration step across time.
     Update accelerations.
     Update velocities.
     Update positions.

Update neighbor relations.
```

The above outline is not designed for speed. The pairwise check of particle interactions if implemented in the most direct manner would be an O(n^2) operation, where n is the number of particles in the system. This would be very slow for n sufficiently large. Likewise the check for collisions with the boundary of the object would be an O(nm) operation, where m is the number of faces of the object. This too would be significantly slow for an object with a large number of faces. To increase the speed of this algorithm we maintain neighbor relations between the particles and restrict ourselves to objects with a small number of (less than 2000) faces. It should be noted the restriction we impose on the number of faces is not that significant as the best our sphere packing can hope to achieve is an approximation of the surface. So unless we are using very small spheres there is not much to be gained in using an excessive number of triangles in the bounding object. Other methods of increasing the performance exist [Palm1995, Gott1996, Kris1998, Mirt1998, Vemu1998, Seni2003].

The methods used to update the acceleration, velocities and position as well as any methods of collision response, should be careful to maintain the momentum of the system. The update of neighbor relations can be incorporated into checks between particles, or excluded. For our implementation it was necessary to maintain. Depending on what stopping criteria is being used, other details may need to be maintained, such as the kinetic energy of the system or a count on the number of particles in collision. Both of which were maintained in our implementation.

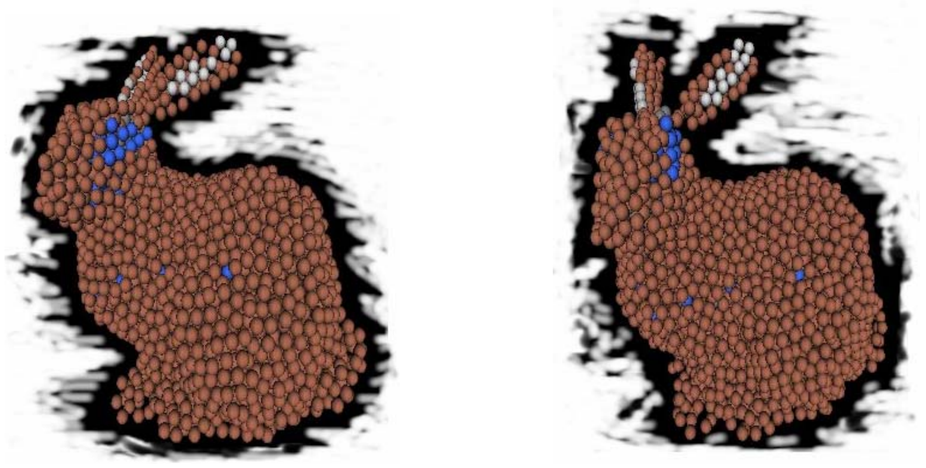

Fig. 5. Stanford bunny filled with 3982 particles, based on a colored smf file.

7 Conclusion and Future Directions

So we have now presented a method for filling a well defined 3D object with particles of a given size. Various results of this method are shown throughout this paper. A possible addition to this method would be the introduction of differently sized or shaped particles. This would allow for greater control of the packing and a wider range of use.

While we have offered no mathematical statement on the density of the packing achieved, we have strong reason to believe it approaches near optimal. Intuitively this can be seen in the hexagonal layering exhibited in the packing. Regardless this method offers a way to represent an arbitrary object as a collection of particles and it does this in such a way as to maintain an approximate representation of the original volume of the object. Further this method allows some control over the density of particles via the forces used in the implementation. In the future this should allow for better investigation into the exact nature of the optimality achieved in the density.

This method should be easily implemented on modeling systems already using particle systems. It should also be capable of being extended to use the GPU of the various graphics cards to increase its performance. In a similar fashion it could be adapted to run in a parallel computing environment. Both of these extensions would be useful tasks to complete in the future.

The method described above offers a variety of possible uses. Specifically for objects represented as particles this method could be adapted to triangulate a dynamically changing surface, assuming the surface particles could be correctly identified. More directly it will allow the application of some particle effects to objects that were not originally particles, such as transforming an object into smoke or water. While this could always be done in a strictly artistic way, it can now be done in a more realistic way that tends to preserve the original volume of the object, and with care the actual energy within the object.

The ability to fill 3D objects may also offer some advantages in other areas. For example various ideas have been proposed in robotic motion planning that demonstrate the use of Voronoi diagrams can simplify the search space for a solution path [Cann1987, Hof1999, Fosk2001]. In a similar manner the arrangement of the particles may allow the (dynamic) computation of the medial axis by slightly modifying ideas presented in other papers [Fosk2003].

So in conclusion we have offered a starting point in obtaining the ability to combine particle representations with other forms of volumetric representations. The method we are using is a physically based method, which should easily fit into any preexisting particle-based modeling system. The packing we achieve should be near optimally dense, but may be modified to allow for other density levels to be achieved. This method will work on most well defined 3D objects, regardless of concavity or holes. It further can be extended to use the GPU of a computer's graphics card or ported to a parallel processing environment for performance gains. This will hopefully allow for the continued development and exploration of particle systems in modeling environments.

Acknowledgements

We would like to thank Dr. John Keyser for the support and advise he has provided us. We would also like to thank Dr. Michael Garland, the University of Illinois and Carnegie Mellon University for making various simple mesh format (SMF) files available on the web, which were used as the "given 3D object" needed to create some of the images in this paper. We also extend thanks to all the unnamed people who created and assisted in the creation of the various SMF files scattered on the web.

Bibliography

[Bara1998] Baraff, Witkin, Large steps in cloth simulation, *SIGGRAPH 1998*.

[Barr2000] Barr, A., M.P. Cani, G., Debunne, M. Desbrun Adaptive simulation of soft bodies in real-time, *Computer Animation 2000 Proceedings*, pp. 15 – 20, 2000.

[Cann1987] Canny, John and Bruce Donald, Simplified Voronoi Diagrams, *A.I. Memo 957*, MIT Artificial Intelligence Lab, April 1987.

[Choi2002] Choi, Kwang-Jin, Hyeong-Seok Ko, Stable but Responsive Cloth, *ACM Transactions on Graphics (TOG), Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 21(3), (SIGGRAPH 2002) July 2002.

[Chri1997] Christensen, J., J. Marks and J. T. Ngo. Automatic motion synthesis for 3D mass-spring models. *The Visual Computer*, 13:20 – 28, 1997.

[Coqu1990] Coquillart, S. Extending free-form deformation: a sculpting tool for 3D geometric modeling. *Computer Graphics Proceedings Annual Conference Series, Proceedings of SIGGRAPH 1990*, pp. 187 – 197, 1990.

[Eber1994] Ebert, D.S., Carlson, W.E., and Parent, R.E., Solid Spaces and Inverse Particle Systems for Controlling the Animation of Gases and Fluids, *The Visual Comp.*, 10:179-190, 1994.

[Fedk2001] Fedkiw, Stam, Jensen, Visual Simulation of Smoke, *SIGGRAPH 2001*.

[Flet1990] Fletcher, C.A.J., *Computational Techniques for Fluid Dynamics*, Springer Verlag, Sydney, 1990.

[Fosk2001] Foskey, Mark, Maxim Garber, Ming Lin, and Dinesh Manocha, A Voronoi-Based Hybrid Motion Planner. *Proc. IEEE/RSJ International Conf. on Intelligent Robots and Systems*, 2001.

[Fosk2003] Foskey, Mark, Ming Lin, and Dinesh Manocha, Efficient Computation of a Simplified Medial Axis. *Proc. ACM Symposium on Solid Modeling and Applications*, 2003.

[Gibs1997] Gibson, S. and B. Mirtich. A Survey of Deformable Modeling in Computer Graphics, *Tech. Report No. TR-97-19*, Mitsubishi Electric Research Lab., Cambridge, MA, November 1997.

[Gott1996] Gottschalk, S., M. C. Lin, and D. Manocha. OOBTree: A hierarchical structure for rapid interference detection, *ACM Computer Graphics (Proc. of SIGGRAPH'96)*, 171–180, 1996.

[Hoff1999] Hoff III, Kenneth E., Tim Culver, John Keyser, Ming Lin, and Dinesh Manocha, Interactive Motion Planning Using Hardware-Accelerated Computation of Generalized Voronoi Diagrams, *Technical Report UNC CS TR99-036*, 1999.

[Kris1998] Krishnan, S. , A. Pattekar, M. Lin, and D. Manocha, Spherical shell: A higher order bounding volume for fast proximity queries, *In Proceedings of WAFR'98*, 287–296, 1998.

[Mirt1998] Mirtich, B., V-Clip: Fast and robust polyhedral collision detection, *ACM Transaction on Graphics*, 17(3):177–208, 1998.

[Muel2002] Müller, M., J. Dorsey, L. McMillan, R. Jagnow, B. Cutler: Stable Real-Time Deformations. *Proceedings of ACM SIGGRAPH Symposium on Computer Animation (SCA)*, pp. 49-54, 2002.

[Palm1995] Palmer, I.J., and R.L.Grimsdale, Collision detection for animation using sphere-trees, *Computer Graphics Forum*, 14(2):105-106, 1995.

[Pfis2000] Pfister, H., M. Zwicker, J. van Baar, M. Gross, Surfels: Surface Elements as Rendering Primitives *SIGGRAPH 2000*.

[Rapa2004] Rapaport, Dennis C. *The Art of Molecular Dynamics Simulation 2nd Edition*, Cambridge University Press, Cambridge, 2004.

[Reev1983] Reeves, W., Particle Systems: A Technique for Modeling a Class of Fuzzy Objects, *Proc. SIGGRAPH 1983*, pp. 359-376, 1983.

[Reyn1987] Reynolds, C., Flocks, Herds, and Schools: A Distributed Behavioral Model, *Proc. SIGGRAPH 1987*, pp. 25-34, 1987.

[Seni2003] Senin, Mikhail, Nikita Kojekine, Vladimir Savchenko, Ichiro Hagiwara, Particle-based Collision Detection, *Short papers proceedings of Eurographics*, EG2003.

[Sims1990] Sims, K., Particle Animation and Rendering Using Data Parallel Computation, *Computer Graphics (SIGGRAPH 1990)*, vol. 24, no. 4, pp. 405-413, 1990.

[Stam1999] Stam, J., Stable Fluids, *SIGGRAPH 1999*, pp. 121-128, 1999.

[Szel1992] Szeliski, R. and D. Tonnesen, Surface Modeling with Oriented Particle Systems, *Computer Graphics (Proc. SIGGRAPH1992)*, vol. 26, no. 2, pp. 185-194, 1992.

[Szel1993] Szeliski, R., D. Tonnesen, D. Terzopoulos, Modeling surfaces of arbitrary topology with dynamic particles. *Proc. Computer Vision and Pattern Recognition Conference (CVPR'93)*, New York, NY, pp. 82-87, June, 1993.

[Terz1988] Terzopoulos, D. and A. Witkin, Physically-based models with rigid and deformable components, *Proc. Graphics Interface*, pp. 146-154, June, 1988.

[Terz1994] Terzopoulos, Demetri, Hong Qin, Dynamic NURBS with geometric constraints for interactive sculpting, *ACM Transactions on Graphics (TOG)*, v.13 n.2, pp.103-136, April, 1994.

[Turk1991] Turk, Greg, Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion, *Computer Graphics (SIGGRAPH 1991)*, Vol. 25, No. 4, pp. 289-298, July 1991.

[Vemu1998] Vemuri, B. C. , Y. Cao and L. Chen, Fast Collision Detection Algorithms with Applications to Particle Flow, *Computer Graphics Forum*, vol. 17, no. 2, pp. 121-, June 1998

[Yngv2000] Yngve, G. D., O'Brien, J. F., Hodgins, J. K., 2000, Animating Explosions. *The proceedings of ACM SIGGRAPH 2000*, New Orleans, July 23-28, pp. 29-36, 2000.