Using Particles for 3D Texture Sculpting

Bedřich Beneš,
Enrique Espinosa,
ITESM Campus Ciudad de Mexico
{beda|eespinosa}@campus.ccm.itesm.mx
http://paginas.ccm.itesm.mx/~beda/research/visual2000.htm

Abstract

Particle systems have been used in computer graphics for many purposes, including visual simulation of fur, grass, hair, and similar fuzzy textures and shapes. The underlying theories used in these algorithms are usually quite complex and are mostly based on simulation of DLA (diffuse limited aggregation), cellular development, reaction-diffusion models, etc. This leads to enormous time complexity. The purpose of this paper is to show that collision detection and distance keeping among moving particles can generate similar realistic textures efficiently. This approach is easy to implement and inherits the major properties from previously published techniques.

Here we first construct scenes consisting of generators of particles, attractors, and cutters - special objects that cause special action of the particle if it is too close. Every particle is oriented and is assigned a table of possible actions that is used for solving critical states. The generators generate particle that is then attracted or repulsed by attractors. When collision with the cutter is detected the particle performs an action according to its state and position in the 3D space. This "intelligent" behavior situates this approach among artificial life algorithms, where the shape is an emerging phenomenon resulting from interacting entities.

Keywords: Computer Graphics, Realistic Image Synthesis, Texture, Particle Systems, Geometric Modeling, Procedural Modeling, and Artificial Life.

1. Introduction and Previous Work

The first application of particle systems in computer graphics can be found in [Reeves83], where they are used for generation of the explosion of a planet for special film trick effects. Two years later [Reeves85] applied similar approach for the generation of huge amounts of data used in simulating forest and grass. In all cases, the plain particle system does not allow any interaction among particles and the environment.

Another application area of particle systems is simulation of plant development. Arvo and Kirk [Arvo88] and Green [Green89] used "intelligent" particles that are able to sense their surrounding environment. They simulate spreading grass, climbing plants, and roots searching for paths on the ground. Later, the particle systems were extensively used for plant simulation and more sophisticated models were introduced, namely interaction of the particles with light e.g., [Benes97]. Flower et al. [Flower92] use interacting particles tight to the surface for spiral phyllotaxis simulation (orientation of seeds, flowers, etc.) in head of sunflowers etc.

Particle systems are also frequently used for generation of fur and similar fuzzy objects. Reaction-diffusion simulation were recognized as a strong tools for texture generation and applied to simulation of color patterns on the fur of zebra, tiger, etc. [Turk92,Witkin92]. Cellular based particle systems simulating reaction-diffusion are used for horny surfaces and for fur simulation in [Fleischer95]. Particles develop under complex conditions that are converted into one differential equation that is then solved. The solution gives orientation and position for every particle.

Another class of approaches for fuzzy object generation includes those based on complex object displaying and rendering. Kajiya [Kajiya89] developed a powerful technique for rendering fuzzy objects allowing even LOD and shadow generation. Perlin [Perlin89] proposed hypertexture as translucent objects above the surface that can be rendered using ray marching algorithms. He also discusses parallel rendering of the particles.
Prusinkiewicz et al. [Prusinkiewicz94] use Open L-systems for visual plant model generation. Their work is a good theoretical and formal framework for interaction of L-systems with an environment.
Looking into the previous work we can see that particles that are used for generation of the complex surfaces are either not interacting at all, or their interaction is based on quite complex conditions that introduces long computational time. The main aim of this article is to show that quite simple conditions, namely collision detection and distance measuring and preservation, can allow us generate surprisingly wide scale of textures.
This article is structured as follows. In the next section, principles of particle systems are explained. Section 3 introduces the generating objects and particle distribution. Section 5 describes complete algorithm for the particle system generation and discusses the special cases. Section 6 focuses the complexity of the algorithm and last section concludes the paper.

2. Particle Systems

Particle is a point element in 3D space that is determined by its position [x,y,z] and orientation given by three vectors [**x**, **y**, **z**]. The orientation can be also defined by unit quaternion **q**, but both representations can be used interchangeably. Quaternion representation is better for avoidance of bending the trajectory of the moving particle and is useful for particle motion simulation. They are used namely if we want to simulate particle paths as branches of plants etc.
A particle system is a set of particles. The main goal of a particle system is simulation of the particle development over time. Animation, or simulation, of particle systems is done in discrete time t_0, t_1,…,t_n. Every discrete time step usually corresponds to one frame of the animation. In many cases we are not interested in animation but only in resulting shape or spatial distribution of particles together with their trajectories, so the simulation is just a tool for obtaining the final shape.
Simulation is consisting of these typical steps.

1. New particles are generated in a distinct space. We use special objects called *generators*.
2. All new particles have an associated initial position and orientation.
3. Position and orientation of all particles in the system (old and new ones) are computed according to external forces and particle-particle or/and particle-environment interaction.
4. Particles that are recognized as too old, too far from the focus of the simulation, etc. are discarded.
5. The system is displayed.

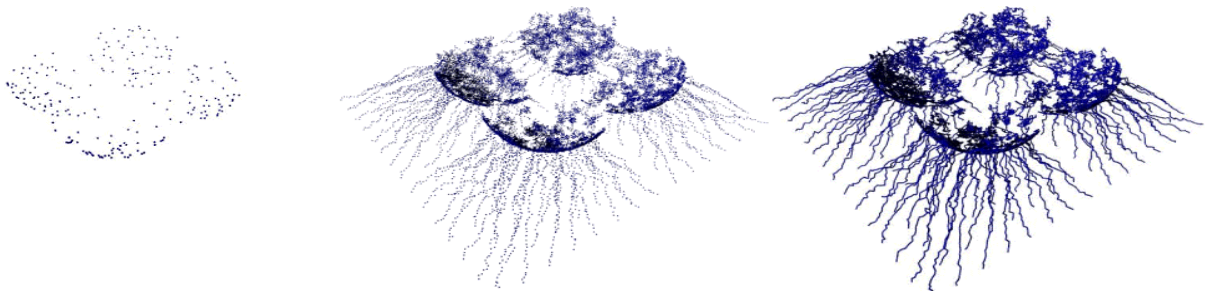


Fig. 1 Particles attracted by four spheres displayed in different modes (left to right): as spheres in the final position, particle trajectories as spheres (middle), and the trajectories as cylinders

It is well known that the important problem underlying particle systems is the last step of every frame calculation – particle system displaying. Particle systems are usually consisting of millions of particles and they can be infinitesimally small. In our case we will display either the particles themselves as spheres or their trajectory as demonstrated in the Fig. 1. However, different techniques can be used as well. The object that corresponds to the particle depends on the simulated phenomena. For example if we want to simulate grass we will display the particle trajectory as a leaf of sedge, for simulation of the flock of mosquitoes point would be sufficient, etc. Most of the previous work focuses on LOD (level of detail) when the particles are displayed. Our technique does not focus on solving the LOD problem. Techniques described in [Fleischer95] can be used here.

3. Generators

Another important factor in simulation of the particle systems is the object that generates the particles. Because we are primarily interested in simulation fur and fuzzy objects we suppose that the object that generates the particles is skin of either human or animal – i.e., B-rep of an object. Previous works [Fleischer95,Turk91,Witkin91] use simulation of reaction-diffusion or diffuse limited aggregation for perfect distribution of particles. This involves solving of differential equations that gives very precise but computationally consuming results. Instead we suggest use of jittering, - approximation of Poisson disc random distribution (see e.g., [Watt92]).

The principle of jittering can be explained as follows (see Fig. 2). First the surface is divided into "equal areas" each associated with one particle in its center. Equal areas are obtained by constant change of the step in the parametric space [u,v], i.e., we supposes parametric surfaces with arc length parametrization that is easy to get. This results in the perfect uniform distribution of the particles. In the second step every particle is shifted within its area by some random number (with equal distribution) in such a way that it remains inside the area. This causes the particles to be distributed randomly, but they cover the entire surface. Jittering is a standard technique used for antialiasing in rendering and it converts aliasing artifacts to visually plausible noise. Fig. 2 demonstrates this technique applied to texturing.

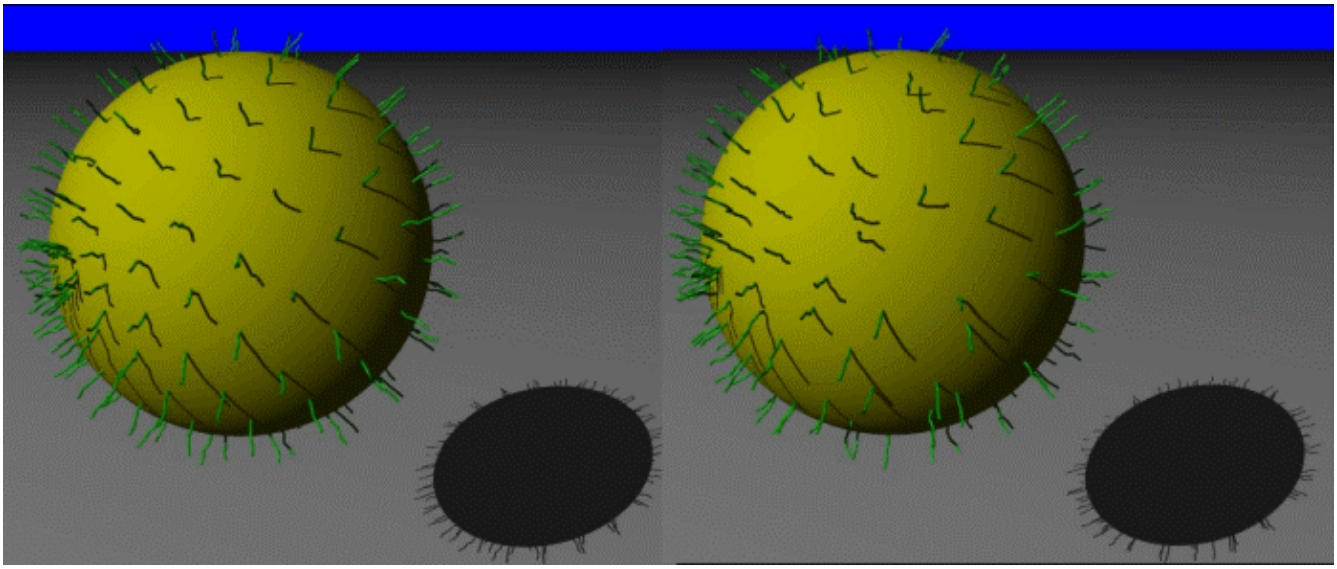
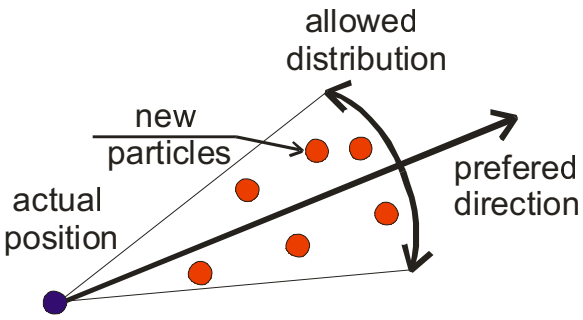


Fig. 2 Regular (left) and jittered distribution of particles on a sphere

The biggest advantage of jittering is that it can be calculated very fast. The random numbers can be stored in a precomputed table and the technique itself is easy and involves only a real number addition. On the other hand it can be complicated if we can use some topologically complex surfaces (Klein's bottle) or such as demonstrated in [Fleischer95]. If the generated object is "well behaved", like plane, sphere, free form surface, etc. then this technique can be used without any particular problems, because the texture mapping function that maps [u,v] texture coordinates to 3D space can be used. In the case of free form surfaces arc length parameterization over both coordinates must be used, but this is well known and already solved problem in CAGD.
Another advantage of the jittering over previously published techniques is that the final textures are usually very complex and therefore the initial position of the texture on the surface is mostly hidden under complex shape.
We must keep in mind that the jittering is simplification, and if we want to use exact representation of the particle distribution on the surface, we should use precise techniques. For most of our applications we have found jittering useful and providing realistic results.

4. Motion of the particles

For the motion of the particles we use *directed random walk* with specified table of actions in special cases (collision, certain distance from the object, etc.).



Random walk is a well-known technique that can be thought of as simulation of Brownian motion [Mandelbrot82]. In the classical random walk we randomly generate a new position for the particle at a certain distance from the actual position. We modify this approach in such a way that we generate *n* random trials and the best, according to the conditions that are described bellow, is chosen. A similar approach was used for simulation of climbing plants as voxel space automata in [Greene89].

Fig. 3 Directed random walk principle

We have not found the idea of *directed random walk* in computer graphics literature. We hereby mean random walk that is determined by certain direction and has limited angle of distribution as shown schematically for 2D case in Fig. 3. This task can be also reformulated as a random walk in polar coordinates with limited angle of distribution.

5. The Algorithm

The complete algorithm for the texture generation that was outlined in previous sections consists of the following steps:

1. Generate initial particles on the surface using jittering
2. Assign initial direction to all particles –randomly perturbed normal to the surface.
3. While the end of the simulation is not reached repeat the following steps
   I. For each particle does:
      a) Generate *n* random positions
      b) For every position evaluate fitness function *f*
      c) Use position with the best *f* value as a new position of the particle

d) Check *critical states* of the particles and perform corresponding actions s

II. Display the system:

We have already discussed the first step of the algorithm. The key issues in the algorithm are steps b) and d), i.e., determining the best position and checking the critical states of the particle. We will discuss them now in more detail.

5.1. The fitness function

The fitness function $f : Real^3 \rightarrow Real$, is a function from the Cartesian 3D space to the space of the real numbers. We can also think about this function as about the field function that associates a real number to every point in the 3D space. The function depends on construction of the scene and on the object behavior. We do not need to evaluate this function for every point in the space, because it can be computed on demand efficiently.

Our task is to minimize this function i.e., to choose the minimum from all the randomly generated positions.

The function f can be chosen arbitrarily, but we have found useful approach similar to electrical charge measuring. Each attractor in the scene has assigned a real number a that corresponds to its ability to attract ($a>0$) or repulse ($a<0$) the particles. The value of the function f

$f_i = a_i \, d,$

where d is the distance between the particle and the i-th attractor. We measure the fitness function for all objects in the scene and the minimum is then chosen. Fig. 4 shows a sphere that attracts (left) and repulses the particles generated on a plane.

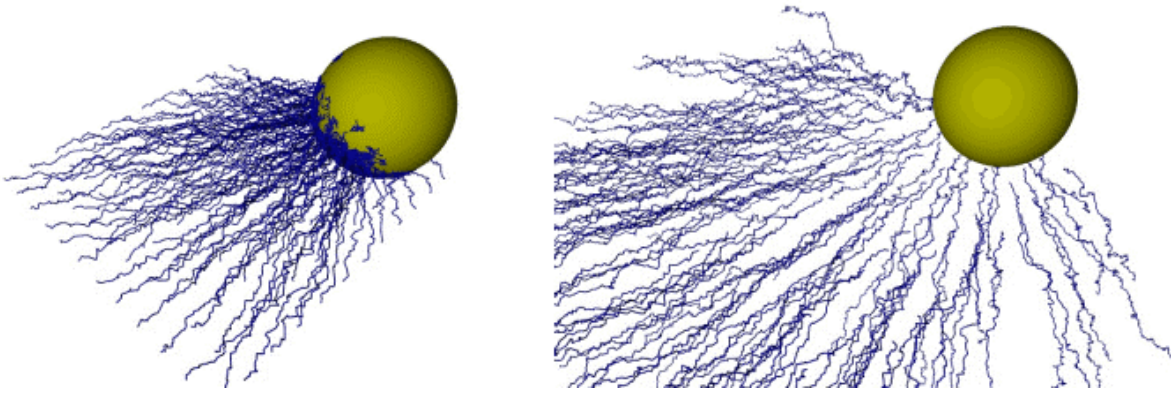


Fig. 4 Positive value (left) attracts the particles, negative (right) repulses.

The value a can change over time i.e., $a(t)$, so the object can change behavior for example as positive/negative pulsar as shown in Fig. 5 (left). Another example shows changing the charge between two pulsars as shown in Fig. 5 (right). Left object has positive value set to one for ten frames, whereas right object has value zero. After ten frames the charges are changed, so that left has zero and the right has value one.

These examples are very simple and they are just showing abilities of this approach. We can go further with the ideas and we can imagine interactive editor of charges with complete continuous edition of curves over time. It would be also useful to limit the influence of the object within certain distance. This would simplify the fitness function evaluation.

It is important to note that this technique is similar to a construction of a 3D scalar field.

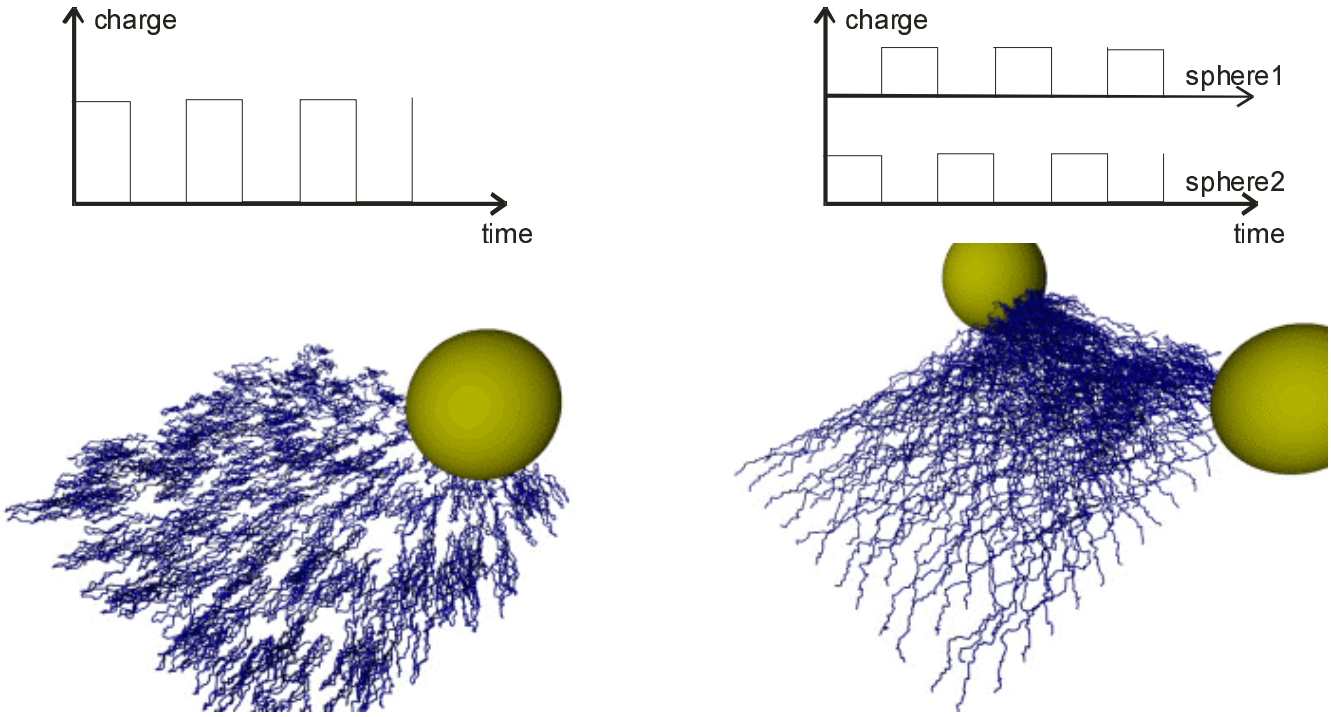
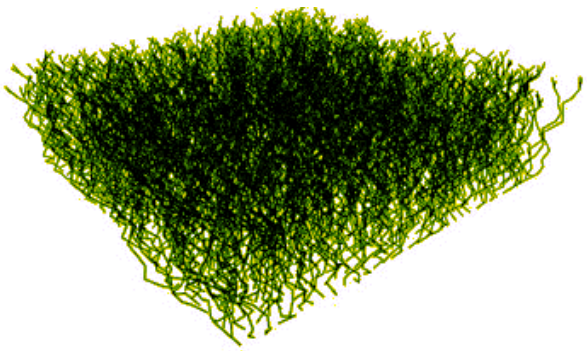


Fig. 5 One attractor (pulsar) changing the charge over time (left), two spheres exchanging the charge over time.

5.2 Critical States

Particles can react to some special situations. We denote these situations by the term *critical states*. We can think about this as about discrete events in continuous simulation (even simulated over discretized time spans) i.e., the particles are moving continuously and sometimes they perform discrete action. We introduce a notion of *cutters.* The cutter is a geometrical object with predefined threshold distance value (implicit surface). When the particle reaches this threshold, special action is performed. Typical action can be: eliminating the particle from the simulation, changing the direction of the motion, etc. The states that we detect are:

- ✓ collision with another particle,
- ✓ collision with an object in the scene,
- ✓ crossing certain predefined distance threshold of a cutter,
- ✓ keeping distance from another particles in the simulation, and
- ✓ age of the particle.



Every critical state has associated a set of actions. For example collision with another particle can cause the run of a physical simulation of bouncing, it can stop motion, it can eliminate the particle from the simulation, etc. The action can be either taken randomly or according to user defined priority.

Fig. 6 Growing grass splits into two particles when crosses certain distance from the ground.

We have found very useful distance measuring and defining action to crossing the distance boundary. The distance condition is very easy to implement and has very important consequences. This is best explained by Fig. 6 that shows growing grass that splits into two streams when crosses certain distance threshold from the plane.
Another example in Fig. 7 shows particles growing from a plane that stop growing at the moment they reach surface of a sphere. This significantly helps reduce computational time of the simulation.

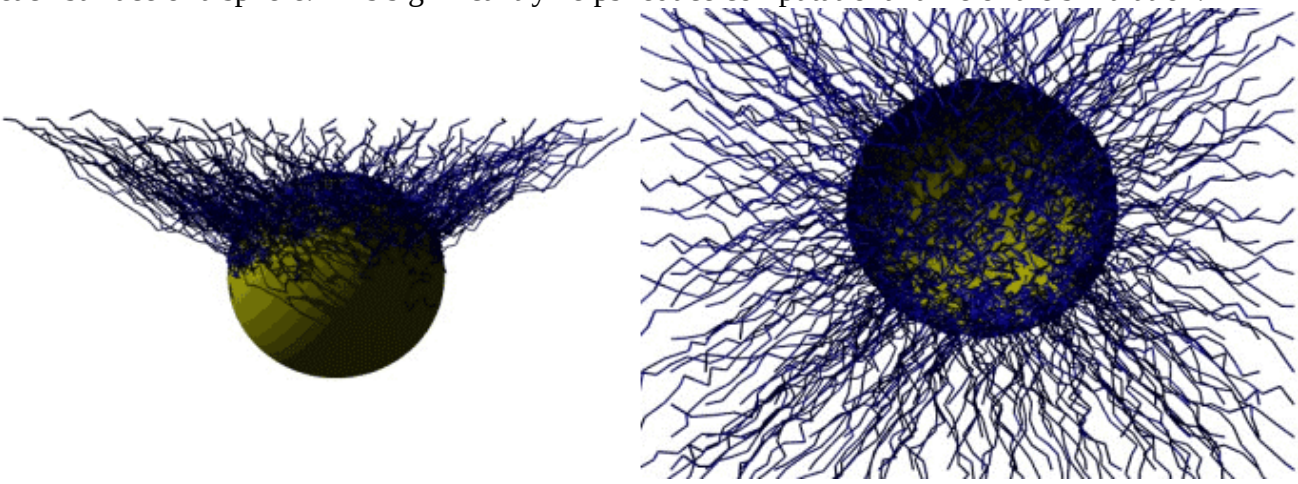


Fig. 7 Particles growing from the plane to sphere are eliminated at the moment they reach the surface (left side and right top view).

5.3 Scene description

The scene is therefore consisting of three different classes of objects. *Generators* are the objects that are responsible for defining the initial particle position and orientation, *attractors* either attract or repulse the particles, and *cutters* cause discrete actions when the particle is too close. One can easily imagine interactive tools for creating the scene that would in a certain way direct the particle generation. We can also extend the idea to building a library of predefined blocks etc.

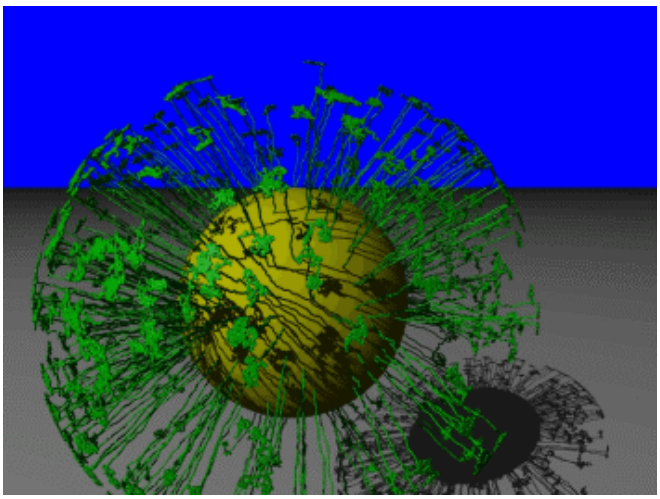


Fig. 8 shows texture generated by the particle system that grows from a sphere to another sphere. The outer sphere is attractor and the cutter. Once the particles reach the outer sphere decreasing counter is set and they are eliminated when the counter is set to zero.

Fig. 8 Particles attracted from a sphere to another one.

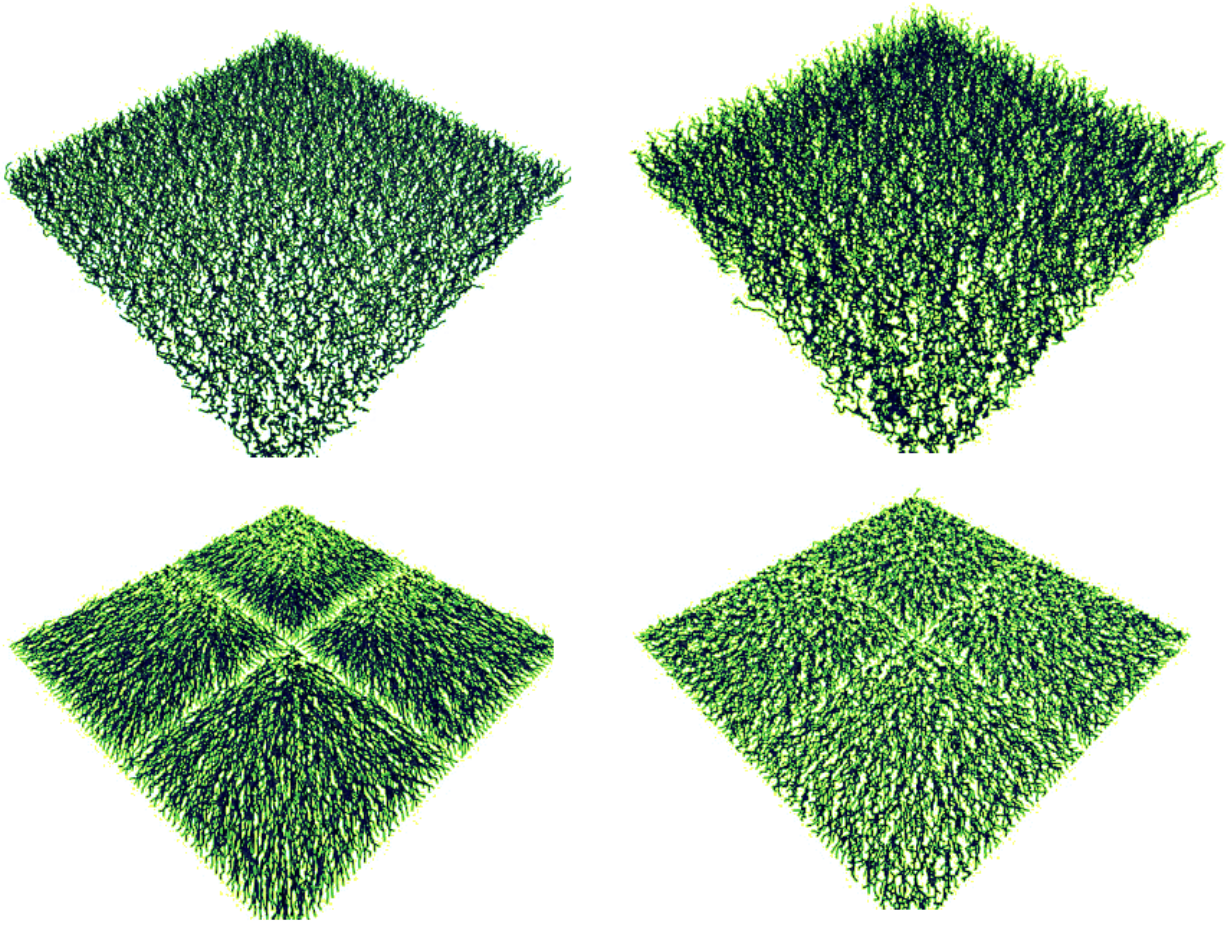


Fig. 9 Grass generated by a random walk with different distance of the plane cutter set (up). Grass growing by the directed random walk attracted by four spheres (as in the Fig.1) with no collision detection (down left) and with collision detection solved.

Another example in Fig. 9 shows texture of growing grass. The grass grows with directed random walk from a plane that acts at the same time as the cutter. Once the particle is too far from the plane it is eliminated from the simulation. On the upper images the particles simply grow in the direction perpendicular to the surface and the cutter with different threshold level set cuts them off. Bottom images show particles growing to four spheres as in Fig. 1. In the bottom left image the particles grow without any specific control, in the right image they growth is controlled by collision detection. In this most complicated case the particle cannot cross the trajectory of another particle and they are also keeping certain distance from each other.

5.4 Collision Detection

There are two kinds of collision detection. The first - collision detection between particle and cutter is easy to solve, because it involves only implicit function evaluation. Here we assume the particle to be a point. The second case – collision between particles is solved by a new technique that is out of the scope of this paper. For this purpose we assume dynamical scene and particles to be spheres.

6. Implementation

The goal of the algorithm that we are proposing here was to create very fast technique that gives visually plausible results. Actually, our algorithm runs interactively on PentiumII/350MHz with OpenGL even up to ten thousand particles without collision detection. The collision detection presents the biggest time demand of this technique. Many acceleration techniques are used in Virtual Reality and robotics for intersection solving and they can be applied immediately here. This was also one of our goals – to propose a technique that is compatible with existing techniques used elsewhere. Anyway fast collision detection presents an open problem and we are working on the solution.

Another pertinent discussion concerns LOD. Our implementation does not provide any LOD control, because it works directly in the object space and does not use any prior information about resolution of the generated image, about camera and its position and orientation, etc. We believe that techniques described in [Fleischer95] can be used here.

7. Conclusions

The technique presented in this paper fast generates 3D textures with realistic appearance. It is a step back from the biologically or physically based algorithms that provide perfect results, but step forward for interactive texture sculpting. The algorithm is based on pure geometrical information about the object distribution in the space where the particles moves and that is why it can benefit from existing techniques for distance measuring, collision detection, etc.

The scene consists on three classes of objects. Generators that generate the particles, attractors, that either attract or repulse the particles, and from special objects - cutters - that are used for controlling the trajectory of the moving particle. The particle changes its trajectory abruptly when reaches the object. The attractors have certain charge that can change in time and they influence the particles with different power. It can be easy to create 3D modeler that will allow interactive manipulation with the objects in the scene. The preliminary results show that our algorithm is sufficiently fast even for large and complex scenes, so it would be also possible to simulate the particle tracing interactively.

Fig. 10 Generator is the statue of Venus and attractors are some points on the same object. On the left image only the texture is displayed whereas on the other the entire object.

On color plates Fig. 10 and Fig. 11 we can see another possibilities of this approach. The first image demonstrates particles that are generated and attracted by different points on the same object. This causes distinct aggregation of particles to appear. Fig. 11 demonstrates the same approach, but the

generator is bounding sphere. In the left image, some points on the Venus attract the particles, whereas in the right image the cutting sphere stops the particles. This causes the points to aggregate but outside the attracting objects. This approach can be thought as a projection of the attractor to the cutting object.
The technique presented in this paper was inspired by artificial life approaches where entities (mobile agents) interact and the shape is an emergent phenomenon given by this interaction.
Another point of view is that this simulation is a dynamical system, i.e., next state of the system depends heavily on the previous one. It should present certain chaotically behavior and from our simulations we can confirm that some particles on the edges between different cutters or attractors behave unpredictable. It is interesting to put a generator exactly on the edge of chaos. The study of particle systems from this viewpoint would be also interesting.
We are deeply indebted to the anonymous referees for help with clarifying the paper.
For animations please visit http://paginas.ccm.itesm.mx/~beda/research/visual2000.htm

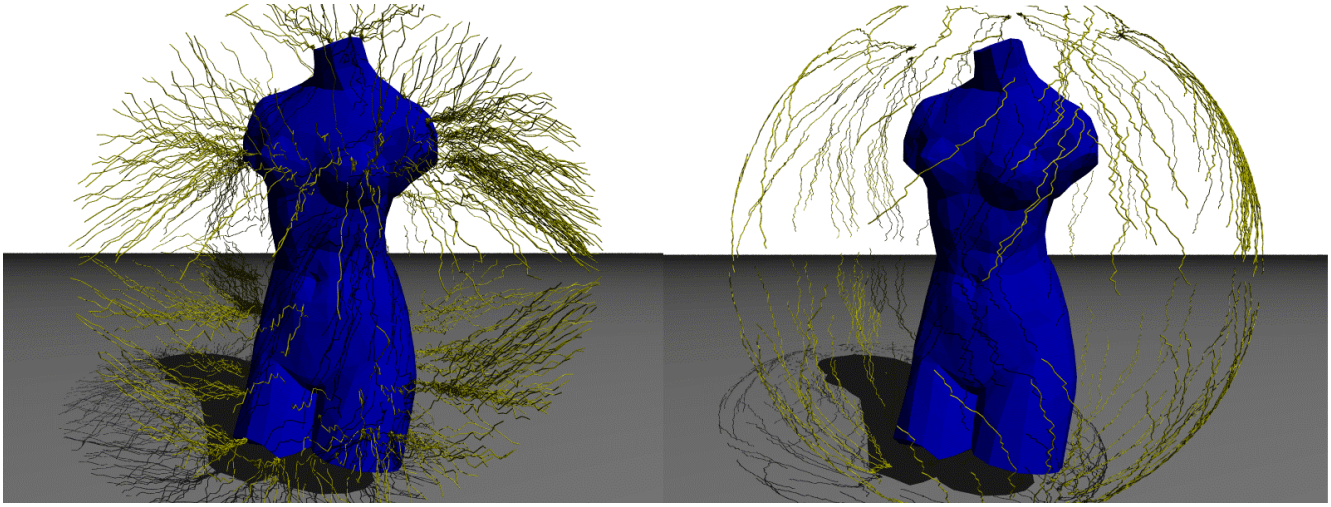


Fig. 11 Particles generated by sphere surrounding the model of Venus are attracted by certain points on the surface. In the left image the particles are allowed to reach the surface, whereas on the right the cutter stops them. This causes aggregation of particles in points that are projection of the attractors to the cutting sphere.

8. References

[Arvo88] Arvo, J., Kirk, D.; *Modeling Plants with Environment-Sensitive Automata*, Proceedings of Ausgraph'88 pp: 27-33, 1988

[Benes97] Beneš, B.; *Visual Simulation of Plant Development with Respect to Influence of Light*, Computer Animation and Simulation'97, Springer--Verlag Wien New York, pp: 125--136, 1997

[Fleischer95] Fleischer, K.W., Laidlaw, D.H., Currin, B.L., Barr, A.H.; *Cellular Texture Generation*, ACM Computer Graphics, Proceedings of Siggraph 1995, pp: 239-248, 1995

[Fowler92] Fowler, D.R., Prusinkiewicz, P. Batjes, J.; *A Collision-based Model of Spiral Phyllotaxis*,

ACM Computer Graphics, Proceedings of Siggraph'92, pp: 361—368, 1992

[Greene89] Greene, N.; *Voxel Space Automata: Modeling with Stochastic Growth Processes*

in Voxel Space, ACM Computer Graphics, Proceedings of SIGGRAPH '89, pp: 175-184, 1989

[Kajiya89] Kajiya, J.T, Kay, T.L; *Rendering Fur with Three Dimensional Textures*, ACM Computer Graphics, Proceedings of Siggraph 1989, pp: 271-280, 1989

[Lewis89] Lewis, J.P; *Algorithms for Solid Noise Synthesis*, ACM Computer Graphics, Proceedings of Siggraph 1989, pp: 263-270, 1989

[Mandelbrot82] Mandelbrot, B.B.; The Fractal Geometry of Nature, W.H.Freeman, San Francisco, 19882

[Perlin89] Perlin, K.; *Hypertexture*, ACM Computer Graphics, Proceedings of Siggraph 1989, pp: 253-262, 1989

[Prusinkiewicz94] Prusinkiewicz. P, James. M, Měch, R.; *Synthetic Topiary*, ACM Computer Graphics, Proceedings of Siggraph 1994, pp: 253-262, 1994

[Reeves83] Reeves, W.; *Particle Systems - A Technique for Modeling a Class of Fuzzy Objects*,

ACM Transaction on Graphics, vol. 2(2), pp: 12—22, 1983

[Reeves85] Reeves, W., Blau, R.; *Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems*, ACM Computer Graphics, Proceedings of SIGGRAPH '85, pp: 313—322, 1985

[Turk91] Turk, G.; *Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion*, ACM Computer Graphics, Proceedings of Siggraph 1991, pp: 289-298, 1991

[Watt92] Watt,A.,Watt, M.; Advanced Animation and Rendering Techniques Theory and Practice, Addison-Wesley, Reading,1992

[Wijk91] van Wijk J.J., A.; *Spot Noise*, ACM Computer Graphics, Proceedings of Siggraph 1991, pp: 309-318, 1991

[Witkin91] Witkin, A., Kass, M.; *Reaction-Diffusion Textures*, ACM Computer Graphics, Proceedings of Siggraph 1991, pp: 299-308, 1991