Collision Avoidance and Surface Flow for Particle Systems Using Distance/Normal Grid

Tommi Ilmonen
Helsinki Univ. of Technology
Telecommunications Software and Multimedia Laboratory
Tommi.Ilmonen@hut.fi

Tapio Takala
Helsinki Univ. of Technology
Telecommunications Software and Multimedia Laboratory
tta@cs.hut.fi

Juha Laitinen
Helsinki Univ. of Technology
Telecommunications Software and Multimedia Laboratory
Juha.Laitinen@tml.hut.fi

ABSTRACT

Fire, explosions, and other special effects are often created with particle systems. In real-time applications the particle systems must be very fast to compute since otherwise the application cannot maintain reasonable frame rate. One part of this challenge is the collision detection between particles and the objects in the scene. We present a new approach to collision detection and surface flow effects for particle systems. In pre-processing phase we rasterize a 3D model into a distance/normal grid. The grid can be used for collision avoidance, to create surface drag and to simulate fluid flow around non-deforming objects. This method is not physically accurate, but it provides visually plausible results. The primary benefit of this method is that it is efficient and its performance is independent of the complexity of the model. This methods works well in real-time, in some cases surpassing the rendering speed of modern graphics hardware by order of a magnitude.

Keywords
Particle animation

1 INTRODUCTION

Particle systems are widely used in both off-line and interactive graphics to simulate fluid phenomena such as fire, explosions, smoke and clouds [Bur00a]. To be fully credible such effects need to interact with their environment.

To handle the collision of a particle with a surface one needs to use either 4D collision detection (taking into account the velocity of the particle) or surround the particle with some geometry (sphere or cube for example) that is used for collision detection. In both cases we must check collision of the particle against all surface primitives – a slow operation if the object is complex.

FULL Papers conference proceedings ISBN 80-86943-03-8
WSCG'2006, January 30-February 3, 2006
Plzen, Czech Republic.

If accurate collision detection is infeasible then one needs to turn to methods that trade accuracy for speed. The traditional approach to collision avoidance has been to identify intersecting objects and then apply collision resolution rules to achieve intersection-free state. Our approach to this problem is to create a combined distance/normal (DN) grid that is used to affect the particles as they fly close to the object. This grid can be used not only for collision avoidance, but also for other fluid effects such as drag and surface flow. many of the artifacts are not visible in the particle animations – the large number of particles and their overall fuzziness hides the imperfections of the physics simulations.

We developed this technique for an interactive particle-system installation where we needed high-performance collision detection. We were also interested in simulating fluid dynamics with the system, since many particle effects represent fluid systems. We concluded that any traditional collision detection system would be too slow for our purposes and a more efficient method was needed. We did not need a physically accurate system but one that is visually convincing. We could leverage the specific features of our installation since the scene was composed of rigid, non-deforming objects. These constraints led us to develop this method.

This paper is organized as follows: First we review existing collision detection systems. Then we introduce the distance/normal (DN) grid and the creation of such grids. After that we go through the effects that one can create with the distance/normal grid – collision avoidance, drag and surface flow. Finally we give some ideas for further development of the method.

2 BACKGROUND

Collision detection is one of the most fundamental problems in computer graphics and much research has been done to create optimal collision detection systems (for a comprehensive survey of techniques,see Hadap et al [Had04a]). While most of the systems handle various kinds of 3D objects there are are also collision detection systems that are specialized to particle systems.

Sims has described the general components of particle systems and also collision detection with spheres and planes [Sim90a]. Karabassi has built a system that performs exact 4D collision detection between particles and solid objects and collision avoidance between particles[Kar99a]. Like many others, her system uses spherical repulsive force fields to keep particles from hitting each other.

Distance fields are frequently used in robotics for collision avoidance. For example Greenspan[Gre96a] and Jung[Jun96a] have used voxel-based method for collision avoidance in robotics. These methods work by storing the distance of a surface from a voxel to the closest surface. In run-time the system checks how far the closest objects are and if there is a risk of collision the systems typically revert to ordinary intersection-based collision detection.

Steele has developed a system that does collision detection, avoidance and response with the aid of a vector field[Ste98a]. This method is similar in principle to our approach, but since Steele's system relies on a few simple geometrical field shapes it cannot be used with more complex objects.

Vector fields have been used since the seventies to visualize fluid flow. In these cases the flow field is first calculated with some physical modeling system and then quantisized into a voxel grid. While the data structures in this approach are almost identical to the distance/normal grid the way the grid is calculated, interpreted and used is different. In the classical vector-grid approach the grid alone determines the path of the particles while we use more complex rules to get collision avoidance and surface flow that are not predefined. Figures 6 and 7 demonstrate how we can create variable effects with one grid – a feature that the older grid-based methods do not support. With our approach it is possible to merge other forces (variable wind, explosions, gravity) with the grid.

3 DISTANCE/NORMAL GRID

The most common way to avoid collisions is to detect and resolve them as they happen. Another method is to use repulsive gradient fields that prevent collisions from happening in the first place. In past, force fields have been used to avoid collision between simple objects like spheres and it has not been possible to create force fields around objects of arbitrary shape. The novelty of our approach lies in the idea of using a distance/normal grid to represent objects of arbitrary shape and a collection of algorithms that can be used for collision avoidance, drag, and surface flow. Compared to previous approaches our system performs faster by taking advantage of both distance and normal information that is stored in the grid. This method works if one of two the colliding objects is a particle – if both were complex objects we would eventually come up with yet another spatial-division collision detection system for rigid bodies.

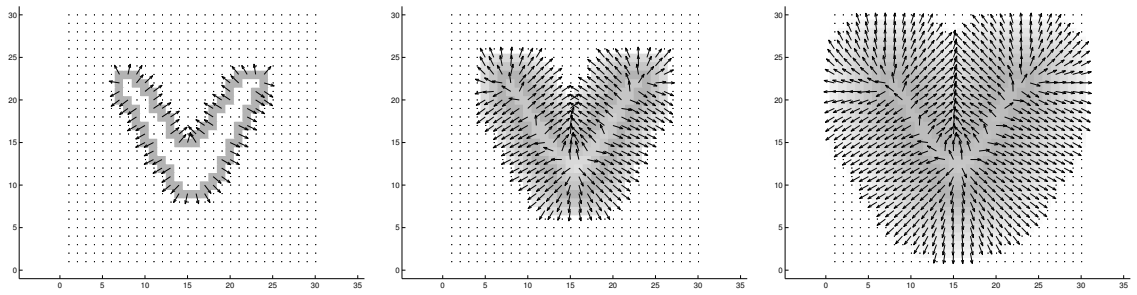
In the grid each voxel contains a three-dimensional unit-length vector (**N**) that indicates the normal of the object and a distance value (l) that tells how far is the closest surface from the center of the voxel. At run-time we can get the distance and normal in any point by looking up the voxel that contains the particle. The computational complexity of the method is O(1) – its run-time performance is not affected by the complexity or shape of the object.

On the general level we first rasterize the object to a 3D grid. Then we thicken the surfaces of the object to cover more voxels and create a DN-grid. The resulting grid is stored to a file. The grid is read from the file in run-time and used with the collision avoidance and surface flow algorithms to make the particle system react to its environment.

3.1 Grid Construction

A critical step in this method is the creation of the DN grid. There are many ways to surround an object with a voxel grid. The basic requirements of the grid are that the normal vectors close to the object should be perpendicular to the surface and the normal vectors should change smoothly outside the surface and follow its overall shape. The grid should be dense enough to capture all relevant details of the object.

We have used two methods to create such grids from polygonal models. The first method is based on first rasterizing the object into the grid and storing surface normal at each voxel. When more than one polygon



(a) Initial distance/normal grid.

(b) Distance/normal grid after one round of blurring (blur kernel with linear attenuation and radius of 3).

(c) Distance/normal grid after three rounds of blurring.

Figure 1: Distance/normal grid around an object (from the same model as in figure 3, with grid resolution of 30x30x30). Each arrow represents the direction of the normal vector **N** at that point and color indicates distance from the surface (darker is closer).

hits the voxel the surface normal is approximated as the average of all normals. We then perform 3D blur on the vectors in the whole grid (exactly analogous to blurring a picture with three color components). We have used small (radius from 2 to 4 voxels) convolution kernels with linear attenuation. The vector grid can be blurred multiple times to smear the direction of the vectors and to spread the field around the object. After each blur round the normals in the voxels that are exactly on the edge of the the objects are returned to their initial value. This step is done to guarantee that the normals are perpendicular to the surface near the object. After the normal vectors have been blurred we normalize them to unit length. At this stage we also set the distance values to indicate distance from the surface. If the voxel is inside the object, then we use negative distance value.

Figure 1 shows a slice of a DN grid that we have created with this method. The arrows show the direction of the field at each voxel and the darkness of the voxel indicates the distance from closest surface. As can be seen this method creates a smooth vector field that follows the shape of the object. By using high-resolution grids we can create DN fields that capture the details of the object but are smooth further away from the object.

This method works usually well but fails to work properly if the object is too thin. In such cases the grid must be extremely dense since the back and front faces of an object must be rasterized to different cells. If the object has near-zero thickness (for example piece of paper) then this method does not work at all – the DN field field will only work towards one direction.

This problem can be addressed with a different method to for calculating the DN grid. With the second method we first rasterize the object into a grid but in each voxel we only store a potential value to indicate that the voxel is occupied. Then we proceed to thicken the potential field by blurring the values (as in previous case). Finally we turn the potential field to a force grid by calculating the normal (gradient) at each voxel with difference method.

The motion of the objects needs to be taken into account when transforming the particle location and velocity from world coordinates to the local object coordinates. Each independently moving object must have a specific DN grid that moves with it.

The objects can also be rotated and translated freely without disturbing the shape of the grid. One can also scale the object uniformly, and the thickness of the DN field will be reduced accordingly. Objects cannot be sheared since shearing changes the angles between the normal vectors and the surface.

When retrieving the value of the DN field in the location of the particle, one can either use the value of the closest voxel to represent both the normal and distance to the closest surface or interpolate these values from the voxels surrounding the particle. We have not noticed any visible difference between these two methods and due to performance reasons we only use the nearest neighbor.

Even though the complexity and shape of a 3D object has no impact on the performance of the system there are objects that do not fit well into this scheme. Objects with very thin extrusions (for example a pencil) can be problematic since the fact that particles are bounced outside the object becomes more visible. A more difficult class of objects are the ones that have multiple thin extrusions that are close to each other (for example trees). In these cases the fields of neighboring extrusions will be merged and this may result

in a field that prevents particles from penetrating the volume of the object even though the object is sparse and the volume could be penetrated.

At run-time the grid is used to represent the object and appropriate rules are used to create bounces, drag and surface flow. The simulation is carried out in discrete time intervals and the rules are applied at each time step. If the grid is too thin then a particle may pass through it without being affected by the field. In practice this implies that the maximum velocity of the particles and the update interval of the physics engine needs to be known when the field is being calculated. While this is an obvious limitation it is not a severe problem since in most cases the creator of the 3D application knows the magnitude of velocities that the particles will have and can adjust the thickness of the field accordingly.

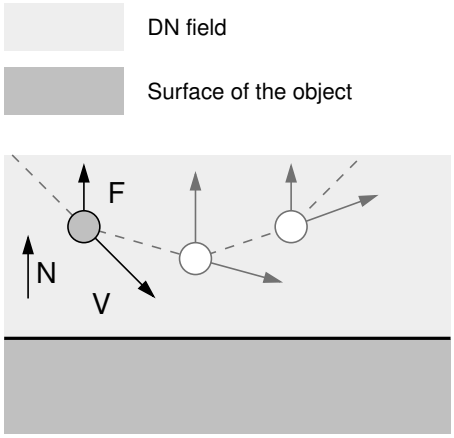


Figure 2: Particles are repelled away from the surface with the spring method. Vector **N** is the surface normal, **V** indicates how far the particle travels during one frame and **F** is the force that is applied to the particle.

3.2 Collision Grid Compression

The grids that we generate are often fairly large. This has a negative effect on performance since it increases the time to load a grid from a disk and easily fills the CPU cache memory. To counter this problem we developed a tree-based compression method.

This method is used as a post-process after the grid has been created. We analyze the grid to find areas where the normal vectors have either not been set or point to similar direction. In these areas the grid nodes are erased and instead a single vector is used to represent all vectors in that node. We also turn the distance/normal -information into a plane equation to avoid problems related to increased cell size. In practice the grid is dense only in areas where the object surface has high curvature.

We have used a two-level tree with tunable division parameters. Increasing the number of levels causes more indirection when accessing tree nodes, but may also reduce memory consumption.

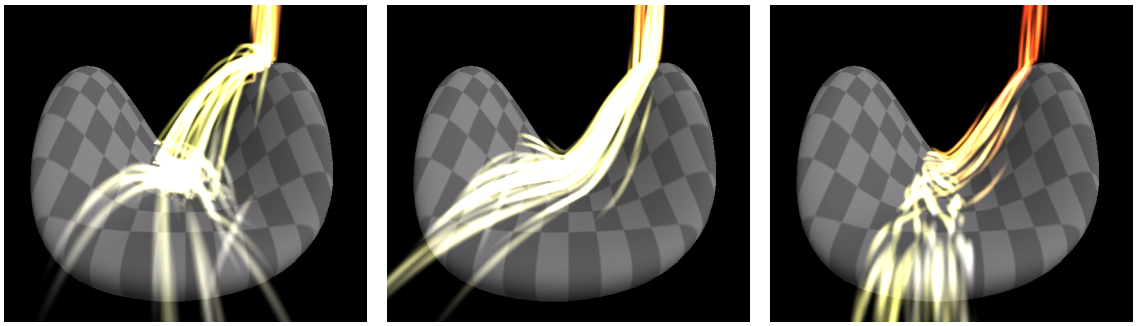
3.3 Collision Avoidance With Spring Method

There are two methods that are useful for avoiding collisions – the spring method and the impact method. The spring method is illustrated in Figure 2. This method works by using the force field as a spring – the deeper the particle passes into the field the more force is applied on it. As a result the particle is thrown back from the surface. The major drawback of this method is that the trajectory of the particle lacks the abrupt change that is caused by impact to a surface. Instead the path of the object is a paraboloid near the surface. Physically this can be understood as an elastic surface that yields as the particle hits it. Problematic artifacts are that a fast particle may fail to bounce from the surface or it may dive temporarily beneath the surface. The exit velocity of the particle is correct if the system is updated with very high processing rate. In practice this is seldom the case, but luckily minor variations in the bounce trajectories make the physics simulations more credible if anything.

3.4 Collision Avoidance With Impact Method

Another method to bounce the particles from a surface is with impact method. With this method we first check if a particle is inside the DN field. If it is, then we use the surface normal and apply direct bounce (or impact) to the particle. This method can be tuned to take into account the velocity of the particle: If the particle is flying with great velocity towards the surface it is bounced as soon as it enters the field. Thus slow particles are bounced close to the surface and fast particles closer or farther away depending on how they to move in the grid as illustrated in Figure 4. The velocity vector V is adjusted to become W after collision with the implied surface.

The impact method is generally superior to the spring method since it causes the physically valid abrupt change in particle velocity and slow particles are bounced close to the surface. In practice one seldom notices that the particles are reflected above or below the surface – the eye does not realize the inexact bounce point of the fast particles and the slow particles are reflected close to the surface as they should. Another factor that helps is that particles are typically rendered as textured billboards, lines or 3D polyhedra. Thus the particle should not bounce at exactly at



(a) Particles collide from a bouncy surface ($\beta = 0.7$).

(b) Particles collide with a non-bouncy slippery surface ($\beta = 0$).

(c) Particles collide with a non-bouncy surface with high drag ($\beta = 0$, $\delta = 2$, $\theta = 1$).

Figure 3: The effect of different bounce and drag properties (60x60x60 grid).

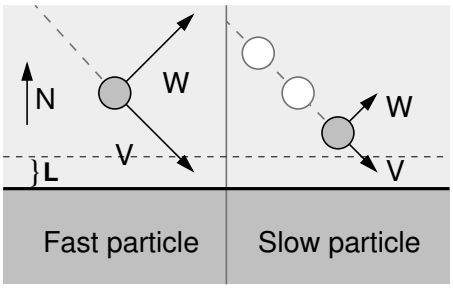


Figure 4: Particles are bounced with the impact method. Fast particles are bounced as soon as they enter the field and slower particles are reflected close to the surface. Vector **W** is the velocity after collision and L is collision threshold.

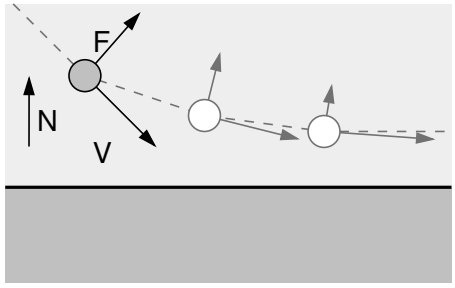


Figure 5: Distance/normal field is used to simulate surface flow by attract a particle to a tangential path.

the surface, but rather above it to take into account the non-zero size of the particle.

3.5 Drag

When two solid objects (or an object and a fluid) interact they usually exert drag. It is easy to do drag calculations with the DN grid. If the particle is closer than a given threshold to the surface then we consider that it is in contact with the surface and drag force is applied on the particle. This method can be combined with the bounce methods since they do not disturb each other. Figure 3 shows combination of using the above impact method for collision resolution and surface drag.

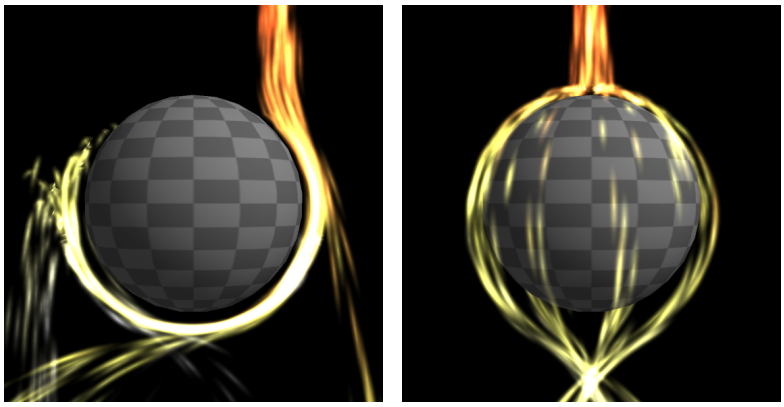
3.6 Surface Flow

Often particles are used to represent fluid effects. In these cases one should use flow equations to determine fluid flow around the objects.

The DN field is useful for implementing surface flow around objects. By surface flow we mean flow close to the object. In this case we apply a force that turns the particle to a trajectory that is tangential to the surface (Figure 5). The force vector **F** should be perpendicular to the velocity vector of the particle. Figures 6 and 7 show the effecs created with this approach.

We have found it is often good to have different coefficients for incoming and outgoing particles (ρ_{in} and ρ_{out}). These parameters can be varied to get different kinds of flows around the object. The following code example corresponds to figure 5 and is also used in figures 6 – 8.

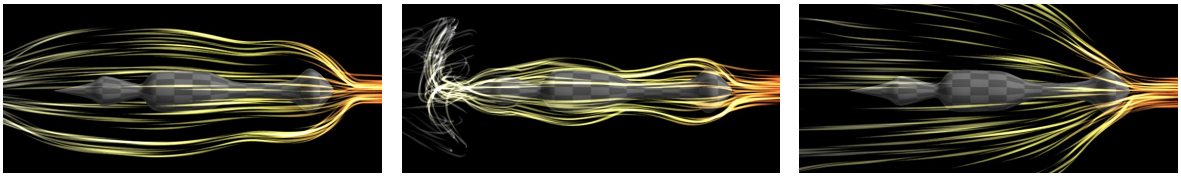
```
γ = N · V
E = (V × N) × V
E_n = E / |E|
if (γ < 0)
F = ρ_in E_n
else
F = ρ_out E_n
fi
```



(a) Particle stream coming to the side of the sphere.

(b) Particle stream falls on top of the sphere.

Figure 6: Particles flow around a sphere (45x45x45 grid). Identical grid is used in both cases.



(a) Strong incoming tangetial force (ρ_{in} = 12) and weak outgoing tangential force (ρ_{out} = 3).

(b) Weak incoming tangetial force (ρ_{in} = 3), strong outgoing tangential force (ρ_{out} = 6) and reduced flow distance. To the left of the shaft one sees turbulence that is caused by excessive tangential grab force.

(c) Weak incoming tangetial force (ρ_{in} = 3) and negative outgoing tangential force (ρ_{out} = -3).

Figure 7: Particles coming from right flow around a shaft (40x40x40 grid). Identical grid is used all cases.

Figure 8 shows the collision avoidance, and surface flow effects together. Collision avoidance is used to keep particles from penetrating the objects and the ground and surface flow is applied to make the particles flow around buildings in a perceptually credible way. This example shows how the DN field can mimic the results of physical systems with high credibility and great performance. In past such effects have been created by evaluating simplified versions of Navier-Stokes -equation which is by several orders of magnitude slower.

This method cannot be used to approximate general fluid-behavior since this method does not address important flow features such as turbulence, edge vortices or colliding fluid streams.

3.7 Performance

The operation of grid-based collision avoidance system is very fast. At run-time we only need to map the location of each particle to corresponding voxel in the grid, retrieve the normal- and distance values from the grid and apply the necessary rules. All of these are fast constant-time operations.

The primary issue for real-time applications is the memory footprint. A 50x50x50 grid that contains four 32-bit floating point numbers per cell takes 2 Mb of memory. A normal PC loads such a grid in 1-3 seconds from the hard disk. If the application must read grids at run-time from a file the few seconds needed to read one grid can be a problem. The grid size also exceeds the cache sizes in most CPUs, resulting in access to the slower main memory of the computer.

Figure 8 is the heaviest simulation in this paper with about 800 particles. Besides the DN field that represents the buildings it has a force generator that pushes particles away from the center of the explosion, gravity and air drag generators. In this system the graphics hardware sets the limits on the performance. An ordinary 1,5 GHz desktop PC can run the dynamics simu-

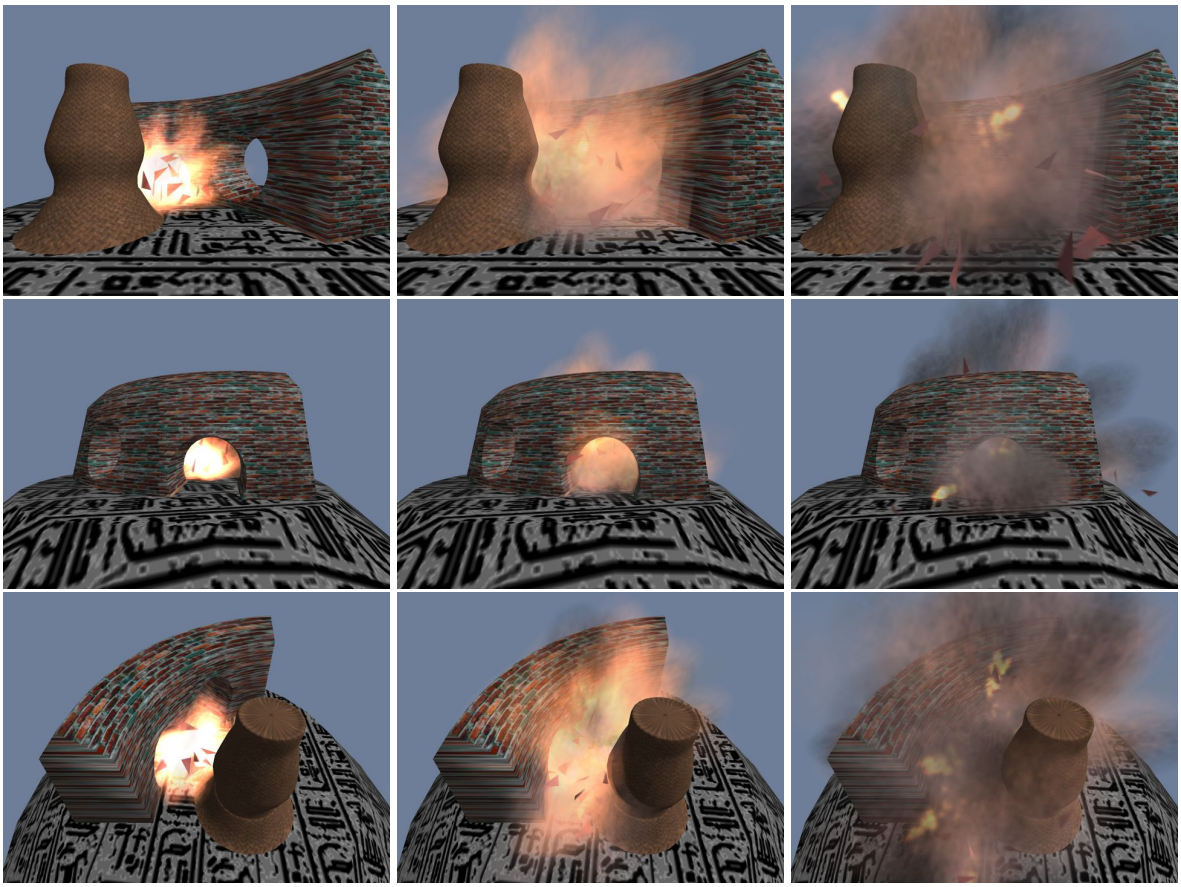


Figure 8: An explosion between objects – each row shows the explosion from one viewing angle at three times (60x60x60 grid, 800 particles). Note how the gas particles flow around the corners of the objects.

Collision System	FPS	FPS loss	Memory usage
None	1800	0	None
Compressed plane grid	1250	550	670kB
Normal grid	950	850	3.7MB
AABB collision detection	450	1350	Not known

Table 1: Performace of different systems in the scene of figure 9.

lation 800 times per second, but the graphics hardware (NVidia GeForce FX 5700) limits the frame rate to 50-80 Hz due to intensive fill-requirements (1024x768 window size, no anti-aliasing).

We have also tested the grid approach against classical collision-detection systems. In this benchmark we used a freely available AABB-based collision library "Opcode", that claims to be a high-performance tool for the task [Ter03a]. We used the ray-triangle intersection test in Opcode. The particle system was a minimal test system that contained simple particles (no color or texture changes) and simple dynamics (only gravity and one collision object). The system was ran without rendering to minimize the effect of rendering on the performance. A snapshot of the scene is in figure 9. The scene was designed to be a difficult case for the collision detection systems — all particles are close to the object and bounding-box -based early-out methods do not work. The test results are in table 1.

As expected the vector grid approach is clearly faster than the classical approach. Surprisingly the compressed vector grid was faster than the normal grid. This may be due to its more simple internal logic (it lacked drag and surface flow calculus) and/or better cache hit ratio.

The grid construction of a 50x50x50 takes a minute or two depending on the parameters (blur radius, unmber of iterations etc.). We have not experimented with optimization or benchamarking of this code since this work is always done in pre-processing phase.

3.8 Implementation

We have implemented the DN field with C++ to a particle engine that already had support for visualization, displaying 3D objects etc. It took 2600 lines of code to add all the features that have been described in this paper. Most of the code is needed to rasterize the 3D object into the grid and to calculate the grid – the collision, drag and surface flow implementations take about 100 lines of code.

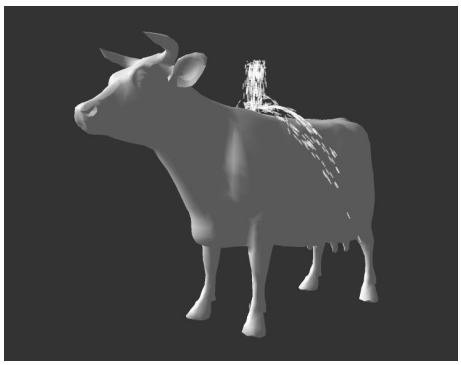


Figure 9: The test scene used in performance comparisons (12 000 triangles).

4 FURTHER DEVELOPMENT

In section 3.1 we presented two methods to generate the DN grid. One could come up with new methods by applying theories that simulate electric fields etc.

At the moment there is a trend to move computations from the CPU to the graphics hardware. The DN grid method might fit to this approach – the field can be stored as a 3D RGBA texture to the graphics hardware. The equations that govern the behavior of particles are very simple, implying that they can be implemented in the graphics hardware. While it is probable that graphics hardware could be used to run the collision detection the possible performance benefits remain to be seen.

5 CONCLUSIONS

The DN grid method can be used in a number of ways to create perceptually valid physics simulations for particle systems. This approach relies on and leverages the features of particle systems. We have shown that the DN field is useful for creating solid-body dynamics (i.e. bounces) and fluid-effects (surface flow and drag) with a single data structure. It is especially suited to situations where several particles are moving in a complex static scene and computational efficiency is an issue. It is not a 4D collision detection system but by using the range information we can create effects beyond pure 3D collision detection.

ACKNOWLEDGEMENTS

This work has been funded by the Academy of Finland.

References

[Bur00a] John van der Burg. Building an advanced particle system. *Game Developer*, March 2000.

[Had04a] Sunil Hadap, Dave Eberle, Pascal Volino, Ming C. Lin, Stephane Redon and Christer Ericson. Collision detection and proximity queries *GRAPH '04: Proceedings of the conference on SIGGRAPH 2004 course notes*, ACM Press, 2004.

[Gre96a] Michael Greenspan and Nestor Burtnyk. Obstacle count independent real-time collision avoidance. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, volume 2, pages 1073–1080. IEEE, 1996.

[Jun96a] Derek Jung and Kamal Gupta. Octree-based hierarchical distance maps for collision detection. In *Proceedings of the 1996 IEEE International Conference on Robotics and Automation*, pages 454–459. IEEE, 1996.

[Kar99a] Karabassi Evaggelia-Aggeliki, Papaioannou Georgios, Theoharis Theoharis, and Alexander Boehm. Intersection test for collision detection in particle systems. *Journal of Graphics Tools*, 4(1):25–37, 1999.

[Sim90a] Karl Sims. Particle animation and rendering using data parallel computation. In *Computer Graphics (Proceedings of ACM SIGGRAPH 90)*, pages 405–413. ACM Press, 1990.

[Ste98a] Kevin L. Steele and Parris K. Egbert. A unified framework for collision detection, avoidance, and response. In *WSCG'98 Conference Proceedings*, volume III, pages 517–524, 1998.

[Ter03a] Pierre Terdiman. Document named Opcode.html located in http://www.codercorner.com/Opcode.htm. Referenced 2.1.2006, Last updated 2003.