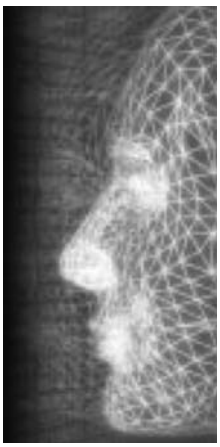
COMPUTER ANIMATION AND VIRTUAL WORLDS
*Comp. Anim. Virtual Worlds* (2009)
Published online in Wiley InterScience
(www.interscience.wiley.com) DOI: 10.1002/cav.287

# Fireworks controller



*By Hanli Zhao, Ran Fan, Charlie C. L. Wang, Xiaogang Jin* and Yuwei Meng*

***This paper presents the fireworks controller, a novel real-time shape-constrained fireworks animation system. We depict the shape of a firework by a 3D mesh. In order to approximate the mesh using evenly distributed points, we propose a fast point sampling method by extending the dual depth peeling algorithm. The samples are then taken as input to shape-constrained fireworks whose physically plausible animations are based on inverse dynamics. We present a highly parallel iterative clustering algorithm to support multi-level fireworks explosion. In order to simulate natural fuzzy fireworks, we impose extra random particles with a parallel random number generator. Several novel intuitive user interfaces are introduced to improve the usability of the system. Experimental results demonstrate the prettiness and efficiency of the proposed approach. Copyright © 2009 John Wiley & Sons, Ltd.***

*Received: 24 March 2009; Accepted: 24 March 2009*

KEY WORDS: particle system; shape constraint; iterative clustering; dual depth peeling; GPU

## Introduction

Computer graphics boasts an amazing success story with regard to realistic simulation. Much work has been done to reproduce the complex natural phenomena and intricate manmade objects using computer graphics. Particle systems have long been recognized as an essential building block for detail-rich and lively visual environments. In particular, the explosion of fireworks can exhibit beautifully orchestrated visual perception. We are often amazed at their fuzziness as well as their level of coordination and synchronization. This is especially true when a bunch of fireworks assume the approximate shape of a real object. We are much delighted to witness the magic art in celebrations, for example, the opening and closing ceremonies of *Beijing 2008 Olympic Games*. In this paper we would like to develop a new system for digitally reproducing such effects. The system has many applications in the entertainment industry. We aim to introduce methods that produce physically plausible fireworks animation, which at the same time, assume a recognizable shape.

A particle system is represented by a large collection of very simple geometric particles that change stochastically over time. The particle system is suitable to mimic dynamic natural phenomena because of its procedural characteristic, that is, new particles are generated in the system and each particle is assigned its individual attributes. All the attributes are updated, and the particles are moved and transformed within each frame. Finally, when a particle in the system passes its prescribed lifetime, it is extinguished. All the living particles are rendered to form a motion sequence. If a particle is constrained with a target shape, its target position is also taken as an attribute.

In this paper, we present a novel real-time physically plausible shape-constrained fireworks simulation system that is capable of satisfying user-specified shape constraints. By extending the dual depth peeling algorithm,[1] we propose a fast point sampling algorithm to evenly approximate the 3D mesh which describes the shape of a firework. These points are then taken as the target positions in the fireworks controller. We control the motion of particles in the firework using inverse dynamics. To make the firework as natural as possible, we introduce multi-level fireworks explosion. We present a parallel iterative clustering algorithm to generate the seeds of the secondary level firework. In order to simulate natural fuzzy fireworks, we impose extra random particles with a parallel random number generator. To facilitate easy user inputs, we introduce several novel intuitive user interfaces to control the shape of fireworks.

In summary, the main contributions of this paper are as follows:

*Correspondence to: X. Jin, State Key Lab of CAD & CG, Zhejiang University, Hangzhou 310027, China.
E-mail: jin@cad.zju.edu.cn

- A novel GPU-based framework of the shape-constrained fireworks simulation system.
- A highly parallel multi-level point sampling algorithm.
- An inverse dynamics based velocity formula for shape-constrained particles.
- A GPU-based stochastic modeling method.
- A set of interesting user interfaces for flexible control.

The rest of the paper is organized as follows. Section "Previous Work" reviews some of the previous work. Section "Pipeline Overview" gives an overview of the pipeline, while Section "Animation Control" describes our animation control algorithms in detail. Section "User Interface" shows the user interfaces we implemented. Experimental results and discussions are presented in Section "Results". Finally, we conclude the paper and suggest some future work in the last section.

## Previous Work

Particle systems have a long history in video games and computer graphics. Early in the 1960s, some games already used 2D pixel clouds to simulate explosions. The use of dynamic particles in computer graphics was first introduced by Reeves.[2] For the modeling of fuzzy objects, he described basic motion operations and basic data representing a particle, both of which have not been altered much since. To improve the performance, Sims[3] developed a particle system on parallel processors of a super computer. McAllister[4] designed many CPU-based velocity and position APIs for particle systems. With the increasing power of modern graphics hardware, some GPU-based systems were implemented.[5,6] However, those papers mainly concentrated upon the general physical properties and rendering techniques. Although Reeves has implemented a spherical generation shape,[2] complicated generation shapes have not yet been addressed. In this paper, we would like to develop a particle animation framework constrained with a target shape. Blythe[7] recently demonstrated a particle system using the powerful geometry shader and stream output. With the geometry shader, the ability to output arbitrary numbers of data to a stream allows the GPU to create new particles and to store the results of computations in the existing particles. In our work, we further exploit the programmability of the GPU to produce complex shape-constrained animation.

A few algorithms have been presented to control the motion of flocks. Reynolds[8] generated complex flocking animation from a few predefined rules. Anderson *et al*.[9] imposed hard constraints on the paths of agents at specific times while retaining the global characteristics of an unconstrained flock. In their method, they generated the animation in a two-stage process. Wojtan *et al*.[10] proposed a keyframe-based framework for controlling systems of interacting particles. They used standard adjoint calculations with gradient-based optimization to control a flocking simulation with several hundred agents. Most recently, Xu *et al*.[11] created a visually pleasing shape-constrained flock animation by a fuzzy control logic. Their fuzzy control logic was designed to dynamically adjust steering forces and control forces.

Some techniques have also been proposed to generate various fluid animations that satisfy constraints and respect some underlying procedural models. Lamorlette and Foster[12] provided both artistic and behavioral control for animation of flames. Treuille *et al*.[13] controlled the smoke simulation by specifying smoke density keyframes. Fattal and Lischinski[14] added two target-driven terms to the standard flow equations for controlling animated smoke. Dobashi *et al*.[15] presented a method for controlling the simulation of cloud formation. Their method can generate realistic clouds while their shapes closely match the user-specified shape.

However, our method is different from those shape-constrained algorithms. Fireworks have their own properties (friction, gravity, explosion, fade away, fuzzy particle, etc.), and therefore those algorithms are not suitable for constraining the animation of fireworks. Our goal is to produce physically plausible shape-constrained fireworks animation in real-time.

## Pipeline Overview

Fireworks, the magic art familiar in celebrations, are suitable to be represented by particle systems. Without loss of generality, our system adopts triangular meshes to control the shapes of fireworks. In the preprocessing step, we first employ an extended dual depth peeling algorithm to produce sample points. In order to support multiple levels of explosion, we then iteratively subsample the higher level points using a parallel clustering method. In addition to subsampled points, another advantage of clustering methods[16] is that we also obtain the correspondence between two consecutive levels once the clustering is completed.

During the animation, we first need to launch new particles into the scene. If these particles are explosive, they will explode and emit new type of particles when





DOI: 10.1002/cav

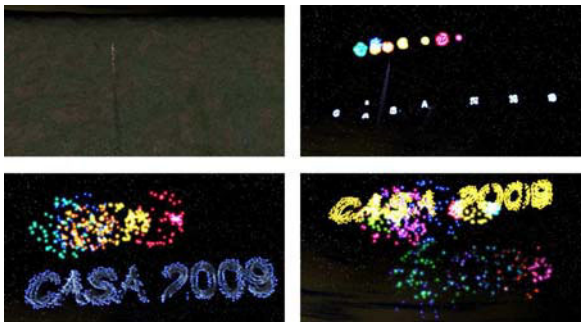



*Figure 1. The process of the "CASA 2009" fireworks animation: (top-left) particle launch, (top-right) first explosion, (bottom-left) formed shape, (bottom-right) another firework.*

they are dead. If they are not, they will fade. We perform the physically plausible animation for shape-constrained particles based on inverse dynamics. In order to simulate natural fuzzy fireworks, extra random particles are also imposed in the scene and the simple Newton's Second Law is taken effect. We observed that the number of streamed out primitives highly limits the parallelism of the geometry shader. Unlike Blythe,[7] we parallelize the animation in the vertex shader while emitting new particles on the multi-core CPU. Figure 1 depicts the process of the "CASA 2009" fireworks animation.

In addition to the animation, the rendering is also important to simulate the real fireworks. In this paper, we render each particle with a texture sprite always facing the active camera like a billboard. As assumed in Reference [2], each particle can be displayed as a point source of light. With this assumption, depth sorting is not required and the amount of light added depends on the particle's transparency and color. Considering that a quickly moving firing object will leave a blurred image and visible dark ashes on the retina of the human eye, we add a special kind of particles which are termed trail particles into the system to simulate the trails of the fireworks. We also use an image or cube map texture as background to improve the visual perception.

# Animation Control

## Point Sampling

The shape of the fireworks is constrained with the input 3D triangular mesh. The simplest way is to get points directly from the vertices of input mesh. If the number of vertices is too small, the density of the constrained particles may be insufficient to form the desired shape. For instance, we cannot control the particles to mimic a cube with only eight vertices. Moreover, the distribution of the original mesh is usually not good. Therefore, we first need to perform the point sampling from the mesh surface. Xu *et al*.[11] used surface mosaic sampling[17] and stratified point sampling methods[18] to impose shape constraints. The sample points spread evenly on the mesh surface. However, both methods are implemented on the CPU and computationally expensive. As a result, they are not suitable for our interactive fireworks controller.

**Algorithm 1** *DualDepthPeeling*(float3 *pos*)

**Require:** Texture2D LDI; int PASS; $BlendOp = \text{MIN}$

```
if PASS > 0 then
    float2 depth = LDI.Load(int3(pos.xy, 0))
    // discard the previously peeled layer
    if (pos.z ≤ depth.x) || (pos.z ≥ −depth.y) then
        discard
    end if
end if
// peel dual layers per pass
return float2(pos.z, −pos.z)
```

In this paper, we propose a GPU-based sampling algorithm which extends the real-time dual depth peeling algorithm[1] to uniformly cover the mesh surface. The depth peeling algorithm was first introduced by Everitt[19] for order independent transparency rendering. The basic dual depth peeling algorithm is described as follows. In the vertex shader, orthogonal projection is set to cover the entire mesh. In the pixel shader, the pseudocode of depth test shown in Algorithm 1 is performed. Occlusion queries are exploited to determine when the Layered-Depth-Image (LDI) becomes empty. The sampling finishes when all depth values are peeled along $x$, $y$, and $z$ axes. To avoid excessive video memory usage, we stream out the sampled points into a vertex buffer. Thus only one LDI texture with two 32-bit floating point channels is needed during the whole sampling process. Since the dual depth peeling algorithm peels two layers per render pass, an almost twice speedup is achieved compared with the traditional depth peeling. Note that the depth peeling-based sampling algorithm is robust even for an unclosed manifold.

After depth peeling, we need to constrain the minimum distance between samples by removing samples that are too close to each other. We can resort to the fast kd-tree based nearest neighbor searching and iteratively remove the samples whose minimum distances to their neighbors are less than a fraction of the peeling



DOI: 10.1002/cav

resolution. Currently, we employ Arya *et al.*'s Approximate Nearest Neighbor library[20] to perform the searching on the CPU. The kd-tree is constructed only once and only a few neighbor points will be found due to the uniformity of the peeling algorithm. Thus the CPU-based library is fast enough.

## Multi-Level Sampling using Clustering

Real world fireworks can explode multiple times to form a target shape. In practice, complex shapes cannot be produced within a single explosion. We use the clustering technique to get decimated low-level shapes. After multi-level sampling using clustering, multiple coarse shapes are constructed and the correspondence between two consecutive levels are established. Iterative clustering algorithms based on Lloyd's algorithm[16] (often referred to as the *k*-means algorithm) have been widely used in computer graphics and computational geometry. Hall and Hart[21] accelerated iterative clustering by using programmable graphics hardware to perform the most computationally expensive portion of the work but the efficiency is very limited. In this paper, we further parallelize the clustering algorithm by exploiting the programmability of modern graphics hardware. Our new clustering algorithm is highly parallel and requires minimal data to be read back to the CPU. In contrast, Hall and Hart performed the fitting and convergence test on the CPU and thus dramatically reduced the parallelism of clustering.

First of all, the vertex buffer containing samples are bound as a buffer texture, their corresponding cluster IDs are stored in an integer texture, and a floating-point depth texture is used to maintain their distances to cluster positions, and cluster position data are stored in another floating-point texture. Since Direct3D does not support reading and writing for the same texture, some auxiliary textures are needed. On current Direct3D 10 hardware, maximum texture size is 8K × 8K.[7] When a point cloud has more than 64M points, more than one textures are needed. In the initialization step, an initial set of seed clusters is created with randomly selected points.

In the partitioning step, we iterate over the clusters. Hall and Hart put the cluster ID into a constant and rendered a quadrilateral once for each cluster. For thousands of clusters, there are a large number of draw calls, state changes, and buffer updates. We can batch the rendering for thousands of clusters using Direct3D 10 instancing. We employ the *SV_InstanceID* system variable to denote cluster ID and thus very few draw calls are invoked during each partitioning. The pixel shader uses the point data and the cluster data indexed by the cluster ID to compute the Euclidean distance. The result is written as the depth value of the pixel, and the cluster ID is written as the color value. We use the depth test to ensure that the cluster ID of the closest cluster for each point is written to the render target. Before writing to the depth buffer, the distance must be scaled to the range [0, 1] using the size of the bounding box of the 3D mesh. Once all clusters have been processed, the depth buffer contains the (scaled) minimum distance for each point, and the render target texture contains the ID of the cluster that produced the corresponding value.

In the fitting step, the proxy position of each cluster is updated. We simply compute mean coordinate of the points in each cluster as its proxy position. This step is inherently a scatter operation since the cluster position depends on a pixel's value in cluster ID texture. We render a point-list with the number of points, while no vertex buffer is bound to avoid extra memory space and transfer time. We rasterize the input pixel position with the *SV_VertexID* system variable. The input position is determined with respect to *SV_VertexID* for each vertex and the texture size (width and height). Then the point coordinates are scattered to their corresponding pixel in vertex shader. The coordinate value and the point number in render target are accumulated with the *addition* alpha blending. Then the mean coordinate is obtained by dividing the point number with a simple quadrilateral draw.

The partitioning and fitting are repeated until convergence. Since the total error is reduced at every step, this algorithm in principle converges to a local minimum, so the termination criterion is simply to repeat until no point moves to another cluster. However, this can take a very long time as most of the improvements happen in the early stages. Also, due to finite precision arithmetic, the algorithm may not actually terminate in practice. Convergence properties of the *k*-means algorithm are described in Reference [16]. In this paper, we compute the convergence criteria in the pixel shader. We first test if the cluster ID for each point is changed. If not, we kill this pixel. Otherwise, we compute $e = (d_i - d_{i-1})/d_{i-1}$, where $d_i$ denotes the (scaled) distance obtained at $i$th iteration. If $e$ is smaller than a user-specified threshold $E$, we also kill it. Then, we employ hardware occlusion query to obtain the number of rendered pixels. If the queried number is smaller than a user-specified number $N$, we consider the clustering to be convergent.





DOI: 10.1002/cav

## Shape Control

In our physical model, we take into account the air friction and the gravity, as well as the mass reduction of burning fireworks. We assume that the mass of a particle linearly decreases with time $t$, that is, $m(t) = M(T-t)/T$, where $M$ is the initial mass and $T$ denotes the lifetime of the particle. In this subsection, we derive the velocity formula for particles whose animations are constrained with a target shape. Our model is based on inverse dynamics. In addition to the external physical forces, the velocity is determined by the target position. Once the correspondence is established, the displacement $S$ from the explosion spot to the sampling point is known. Here we assume the particle spends time $T_0 \leq T$ to move to the target position, and the friction coefficient $k$ and the gravity coefficient $g$ are constant.

Let $f$ be the external force applied to the particle, $a$ be its acceleration, and $v$ be current velocity. According to the Newton's Second Law, both the gravity and the friction control the velocity of a particle, that is

$$f = ma = m\frac{\mathrm{d}v}{\mathrm{d}t} = -kv + mg$$

$$M\frac{T-t}{T}\frac{\mathrm{d}v}{\mathrm{d}t} = -kv + \frac{T-t}{T}Mg$$

$$\frac{\mathrm{d}v}{\mathrm{d}t} + \frac{kT}{M(T-t)}v = g$$

By using the variation of constants method, the following equation can be obtained:

$$v = \left(C + \frac{Mg}{kT-M}(T-t)^{1-\frac{kT}{M}}\right)(T-t)^{\frac{kT}{M}}$$

$$v = C(T-t)^{\frac{kT}{M}} + \frac{Mg}{kT-M}(T-t) \quad (1)$$

Once the displacement constraint takes effect, we can compute the coefficient $C$ as follows:

$$S = \int_0^{T_0} v\mathrm{d}t = C\int_0^{T_0}(T-t)^{\frac{kT}{M}}\mathrm{d}t + \frac{Mg}{kT-M}\int_0^{T_0}(T-t)\mathrm{d}t$$

$$C = \left(S - \frac{Mg(2TT_0 - T_0^2)}{2(kT-M)}\right)\frac{\frac{kT}{M}+1}{T^{\frac{kT}{M}+1} - (T-T_0)^{\frac{kT}{M}+1}} \quad (2)$$

As a result, the velocity component in $y$ axis is solved using Equations (1) and (2). Note that if $|kT - M| < \varepsilon \leq 0.0001$, we should add a small $\varepsilon$ to avoid division by zero. However, there is no gravity in horizontal directions ($x$ and $z$ axes) and only the friction affects the particle's animation. Thus the velocity components in $x$ and $z$ axes can simply be solved using the following equations:

$$v = C(T-t)^{\frac{kT}{M}} \quad (3)$$

$$C = S\frac{\left(\frac{kT}{M}+1\right)}{T^{\frac{kT}{M}+1} - (T-T_0)^{\frac{kT}{M}+1}} \quad (4)$$

From Equations (2) and (4), we can see that the coefficient $C$ is constant and can be calculated once a new shape-constrained particle is emitted. The shape-constrained velocity at time step $t$ can be numerically calculated according to $C$, whereas the velocity for a natural particle is trivially computed with the Newton's Second Law.

## Stochastic Modeling

In order to simulate natural fuzzy fireworks, we need to impose extra random particles on the scene. As discussed in Reference [2], stochastic processes that randomly select each particle's appearance and movement are constrained by a set of parameters. In this paper, we use a parallel random number generator to control the number of new particles, the initial velocities, and the initial colors.

**Algorithm 2** *SampleNoise*(uint4 *texCoord*, uint *key*)

**Require:** uint4 DIGEST; uint ROTATE[4]; uint SINE[16]

```
uint data[16] = SetupInput(texCoord, key)
uint4 d = DIGEST
[unroll] 16-round data compression
for i = 0 to 15 do
    d = uint4(d.yzw, d.y + LeftRotate((d.x +
    F(d.yzw) + data[i] + SINE[i]), ROTATE[i%4]))
end for
return DIGEST + d // add the magic initialization
constant
```

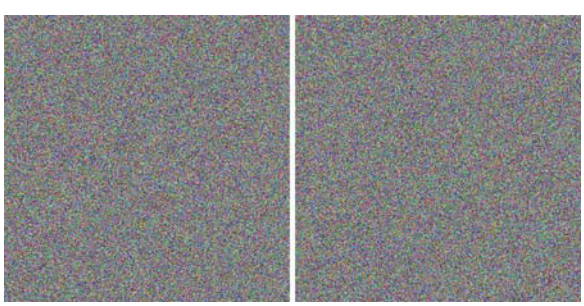


*Figure 2. White noise generation using (left) Tzeng and Wei's full version and (right) our reduction version, respectively.*





DOI: 10.1002/cav

In order to simulate fuzzy objects on the GPU, a good parallel random number generator is essential. Recently, Tzeng and Wei[22] implemented a random number generator in the pixel shader based on MD5 cryptographic hash functions whose statistical robustness has been examined under heavy scrutiny by cryptologists. This generator allows us to compute random numbers in parallel just like ordinary texture fetches: given a texture coordinate, instead of returning a texel as in ordinary texture fetches, the generator computes a random noise value based on the given texture coordinate.

As shown in Algorithm 2, we do a few additional optimizations in this paper. Unlike Tzeng and Wei, who computed the sine function on the fly and rotated the ROTATE vector explicitly, we store the values of sine functions in the high-speed constant registers

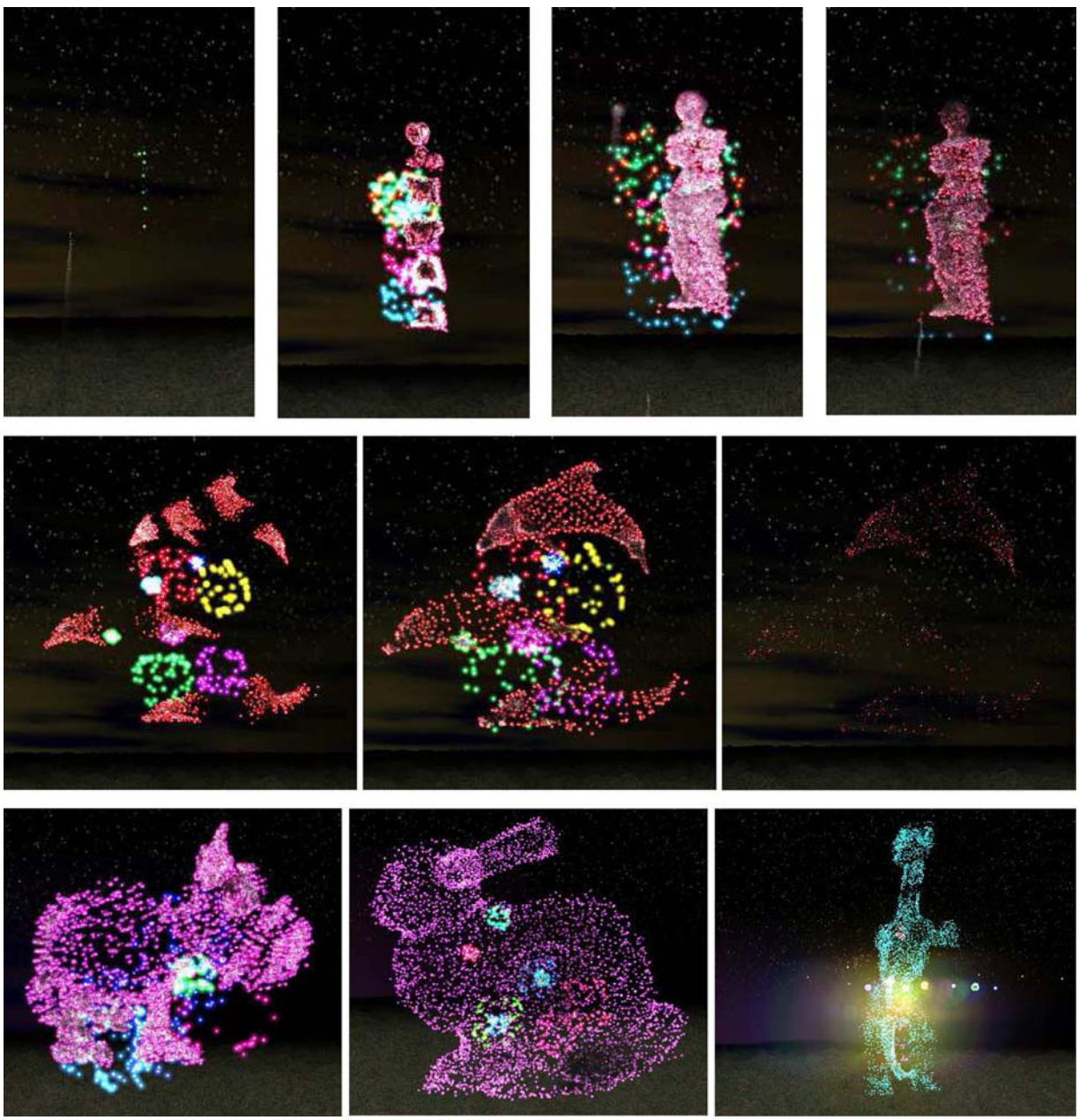


*Figure 3. Various shape-constrained fireworks.*





DOI: 10.1002/cav

| | | 1024 Resolution | | 256 Resolution | | 64 Resolution | |
|---|---|---|---|---|---|---|---|
| Input mesh | Face/vertex number | Samples | Timings | Samples | Timings | Samples | Timings |
| Bunny | 15 000/7502 | 650 358 | 4.372 | 52 073 | 0.271 | 4316 | 0.019 |
| Dragon | 202 520/100 207 | 1 031 122 | 5.084 | 81 381 | 0.296 | 5540 | 0.036 |

**Table 1. Performance statistics of the proposed sampling method with different peeling resolutions. The timings are in second**

and implicitly rotate the ROTATE vector with the *modulus* operator. In the original algorithm, 64 rounds of compression are required to generate numbers that are random enough. In this paper, we reduce it to 16 rounds in order to further improve the speed. We observed that the quality of this reduction version reported in their paper is still acceptable for us (see Figure 2).

## User Interface

Our system supports several intuitive user interfaces to provide flexible control for users. In addition to the simulation effect, the freedom of customization can improve the user experience.

*Mesh input*: the basic input to the system is the triangular mesh. Users can freely import a mesh to the scene and change its position by mouse dragging.

*Sketch input*: we have incorporated Igarashi *et al.*'s sketch-based modeling tool[23] in our system. It is an easy-to-use interface even for children. They can create interesting shapes easily and watch the fireworks forming the shape they draw interactively.

*Character input*: the system currently supports numbers and characters. We have created a 3D mesh database for all characters and numbers. After the user input the character string, the system will create the corresponding meshes by indexing. Users can also input words using voices to free them from mouses and keyboard. We use IBM ViaVoice® library to recognize the speech.

*Animation recording*: users expect to record their customized fireworks animations that can be shown to their friends. The video encoding is performed on the CPU in a separate thread that runs in parallel with the GPU.

## Results

We have tested the system on some scenes in order to evaluate its effectiveness and efficiency. All tests were conducted on a 2.40 GHz Intel Core 2 Quad Q6600 CPU with an NVIDIA GeForce 8800 GTS (512 MB) GPU. With the modest graphics hardware, the frame rates of our system vary from 20 to 100 frames per second for tens of thousands of particles in a scene. The accompanying video is provided to demonstrate the effect of the system. We show various interesting results of the shape-constrained fireworks in Figure 3.

We show the performance statistics of our sampling method in Table 1. For a given mesh, the method is in squared time complexity with respect to the LDI resolution. On the other hand, the sample number also increases with the LDI resolution. Thus the timing is linear with respect to the sample number. Figure 4 shows an example of the sampling result under 256 resolution. Note that contours are somewhat highlighted because of the *addition* alpha blending. A top view with zooming in is

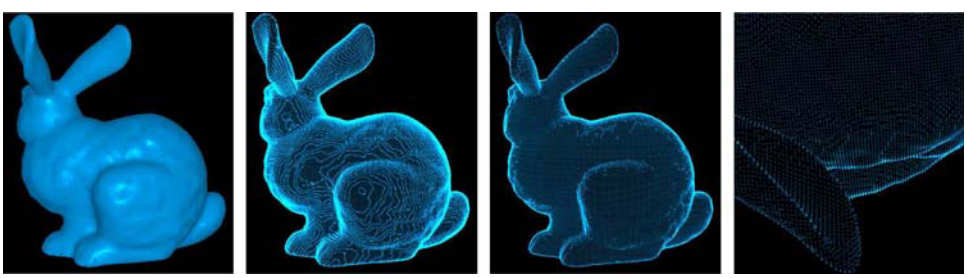


*Figure 4. Sampling result under 256 resolution: (leftmost) input mesh, (mid-left) after depth peeling, (mid-right) minimum distance removal, and (rightmost) top view with zooming in.*



DOI: 10.1002/cav

| Point cloud | Input number | Timings using 1 thread | Timings using 4 threads | Timings on the GPU |
|---|---|---|---|---|
| Buddha | 1 744 018 | 140.061 | 36.746 | 7.456 |
| Dragon | 2 209 226 | 177.466 | 46.581 | 9.425 |
| Flower | 2 234 694 | 179.572 | 47.065 | 9.547 |

**Table 2. Experimental comparisons show the efficiency of our GPU-based clustering. All point clouds are clustered into 10 000 points. The timing results are in second per iteration**

posted in the leftmost image for clarification. We can see that the samples cover the bunny mesh well.

We have compared our GPU-based clustering method with the CPU-based implementation. Since both algorithms compute the same result and perform the same number of iterations, we show in Table 2 how long each iteration of the algorithms takes. Using our GPU-based acceleration technique, we are able to achieve five times as fast as the CPU-based implementation executing in four parallel threads. It is somewhat surprising that the GPU offers such a significant speed improvement over the CPU. We also show the clustering results in Figure 5 for illustration.

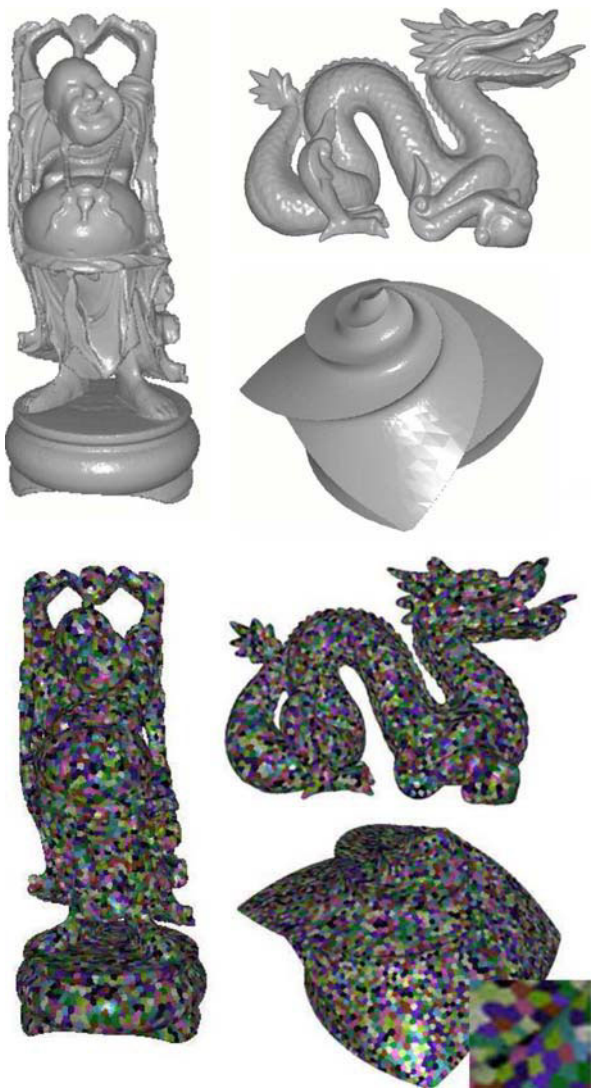


*Figure 5. Test examples: (top) input point clouds and (bottom) clustering results.*

## Conclusion

In this paper, we have proposed a novel GPU-based shape-constrained fireworks simulation pipeline. The dual depth peeling based point sampling algorithm can sample one million points with about 5 seconds while providing good distribution. The parallel iterative clustering algorithm outperforms the CPU-based implementation with several times of speedup. The animations are physically plausible, fuzzy enough, real-time, and easy to use.

In the future, we plan to implement the kd-tree construction on the GPU using CUDA[24] and integrate it in our system. However, the tree depth is highly dependent on the video memory. Currently, we only use the centroid coordinates as the cluster positions. We expect a better scheme[25] to position the clusters. Since our GPU-based multi-level point sampling method is general, we believe that the proposed method can be applied to various point-related and clustering-based applications. As animation and rendering are separable steps, we can easily import our fireworks controller to existing rendering engines for better visualization.

#### ACKNOWLEDGEMENTS

The authors thank Xiaoyan Luo for her kind help in presenting the manuscript and the AIM@ SHAPE shape Repository for sharing the models used in this paper. This work was supported by the National Key Basic Research Foundation of China





DOI: 10.1002/cav

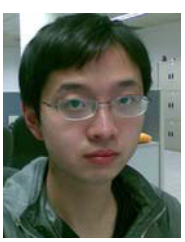
(Grant No. 2009CB320801), the National Natural Science Foundation of China (Grants Nos. 60533080 and 60833007), and the Key Technology R&D Program (Grant No. 2007BAH11B03).

## References

1. Bavoil L, Myers K. Order independent transparency with dual depth peeling. *NVIDIA OpenGL SDK*, 2008.
2. Reeves WT. Particle systems—a technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics* 1983; **2**(2): 91–108.
3. Sims K. Particle animation and rendering using data parallel computation. *Proceedings of ACM SIGGRAPH'90*, 1990; 405–413.
4. McAllister DK. The design of an API for particle systems. *UNC Computer Science Technical Report*, January 2000.
5. Kipfer P, Segal M, Westermann R. UberFlow: a GPU-based particle engine. *Proceedings of ACM SIGGRAPH/EUROGRAPHICS Conference Graphics Hardware'04*, 2004; 115–122.
6. Kolb A, Latta L, Rezk-Salama C. Hardware-based simulation and collision detection for large particle systems. *Proceedings of ACM SIGGRAPH/EUROGRAPHICS Conference Graphics Hardware'04*, 2004; 123–131.
7. Blythe D. The Direct3D 10 system. *Proceedings of ACM SIGGRAPH'06*, 2006; 724–734.
8. Reynolds CW. Flocks, herds and schools: a distributed behavioral model. *Proceedings of ACM SIGGRAPH'87*, 1987; 25–34.
9. Anderson M, McDaniel E, Chenney S. Constrained animation of flocks. *Proceedings of ACM SIGGRAPH/EUROGRAPHICS Symposium Computer Animation'03*, 2003; 286–297.
10. Wojtan C, Mucha PJ, Turk G. Keyframe control of complex particle systems using the adjoint method. *Proceedings of ACM SIGGRAPH/EUROGRAPHICS Symposium Computer Animation'06*, 2006; 15–23.
11. Xu J, Jin X, Yu Y, Shen T, Zhou M. Shape-constrained flock animation. *Computer Animation and Virtual Worlds (CASA'08)* 2008; **19**(3–4): 319–330.
12. Lamorlette A, Foster N. Structural modeling of flames for a production environment. *Proceedings of ACM SIGGRAPH'02*, 2002; 729–735.
13. Treuille A, McNamara A, Popović Z, Stam J. Keyframe Control of smoke simulations. *Proceedings of ACM SIGGRAPH'03*, 2003; 716–723.
14. Fattal R, Lischinski D. Target-driven smoke animation. *Proceedings of ACM SIGGRAPH'04*, 2004; 264–270.
15. Dobashi Y, Kumumoto K, Nishita T, Yamamoto T. Feedback control of cumuliform cloud formation based on computational fluid dynamics. *Proceedings of ACM SIGGRAPH'08*, 2008; article no. 94.
16. Lloyd SP. Least square quantization in PCM. *IEEE Transactions on Information Theory* 1982; **28**(2): 129–137.
17. Lai YK, Hu SM, Martin RR. Surface mosaics. *The Visual Computer* 2006; **22**(9–11): 604–611.
18. Nehab D, Shilane P. Stratified point sampling of 3D models. *Proceedings of EUROGRAPHICS Symposium Point Based Graphics'04*, 2004; 49–56.
19. Everitt C. Interactive order-independent transparency. *NVIDIA Technical Report*, 2001.
20. Arya S, Mount DM, Netanyahu NS, Silverman R, Wu A. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM* 1998; **45**(6): 891–923.
21. Hall JD, Hart JC. GPU acceleration of iterative clustering. *Proceedings of ACM Workshop on General Purpose Computing on Graphics Processors and Poster of SIGGRAPH'04*, 2004.
22. Tzeng S, Wei LY. Parallel white noise generation on a GPU via cryptographic hash. *Proceedings of ACM International Symposium Interactive 3D Graphics and Games'08*, 2008; 79–87.
23. Igarashi T, Matsuoka S, Tanaka H. Teddy: a sketching interface for 3D freeform design. *Proceedings of ACM SIGGRAPH'99*, 1999; 409–416.
24. Zhou K, Hou Q, Wang R, Guo B. Real-time kd-tree construction on graphics hardware. *Proceedings of ACM SIGGRAPH Asia '08*, 2008; article no. 126.
25. Lee CH, Varshney A, Jacobs DW. Mesh saliency. *ACM Transactions on Graphics* 2005; **24**(3): 659–666.

*Authors' biographies:*

**Hanli Zhao** is a Ph.D. candidate of the State Key Lab of CAD & CG, Zhejiang University, China. He received his B.Eng. degree in software engineering from Sichuan University in 2004. His research interests include non-photorealistic rendering, image processing, and general-purpose GPU computing. Contact him at hanlizhao@gmail.com.



**Ran Fan** is a Ph.D. candidate at the State Key Lab of CAD & CG, Zhejiang University, China. He received his B.Sc. degree in mechanical engineering form Zhejiang University. His research interests include digital geometry processing and crowd animation. Contact him at fanran1029@gmail.com.





DOI: 10.1002/cav

**Charlie C. L. Wang** is currently an assistant professor at the Department of Mechanical and Automation Engineering, the Chinese University of Hong Kong. He gained his B.Eng. (1998) in mechatronics engineering from Huazhong University of Science and Technology, M.Phil. (2000) and Ph.D. (2002) in mechanical engineering from the Hong Kong University of Science and Technology. He is a member of IEEE and ASME. His current research interests include geometric modeling in computer-aided design and manufacturing, biomedical engineering, and computer graphics, as well as computational physics in virtual reality. He has received a few awards including the best paper award of ASME 21st and 28th Computers and Information in Engineering Conference (CIE) in 2001 and 2008, and the exemplary teaching award from the Faculty of Engineering (CUHK) in 2007 and 2008. Contact him at cwang@mae.cuhk.edu.hk.

**Xiaogang Jin** is a professor of the State Key Lab of CAD & CG, Zhejiang University, China. He received his B.Sc. degree in computer science in 1989, M.Sc. and Ph.D. degrees in applied mathematics in 1992 and 1995, all from Zhejiang University. His current research interests include implicit surface computing, special effects simulation, mesh fusion, texture synthesis, crowd animation, cloth animation, and non-photorealistic rendering. Contact him at jin@cad.zju.edu.cn.

**Yuwei Meng** is a Ph.D. candidate of the State Key Lab of CAD & CG, Zhejiang University, China. He received his B.Sc. degree in software engineering from Wuhan University of Technology in 2006. His research interests include computer graphics and physically based simulation. Contact him at mengyuwei@cad.zju.edu.cn.





DOI: 10.1002/cav