Computational Visual Media
DOI 10.1007/s41095-015-0025-1 Vol. 1, No. 4, December 2015, 351–358

Research Article

GPU based real-time simulation of massive falling leaves

Chengyang Li[1], Jingye Qian[1], Ruofeng Tong[1](✉), Jian Chang[2], and Jianjun Zhang[2]

Abstract As an important autumn feature, scenes with large numbers of falling leaves are common in movies and games. However, it is a challenge for computer graphics to simulate such scenes in an authentic and efficient manner. This paper proposes a GPU based approach for simulating the falling motion of many leaves in real time. Firstly, we use a motion-synthesis based method to analyze the falling motion of the leaves, which enables us to describe complex falling trajectories using low-dimensional features. Secondly, we transmit a primitive-motion trajectory dataset together with the low-dimensional features of the falling leaves to video memory, allowing us to execute the appropriate calculations on the GPU.

Keywords real-time simulation; falling leaves; GPU acceleration

1 Introduction

Falling leaves, as a universal feature of autumn, are often shown in movies and games. Simulation of falling leaves has been studied for many years in hydrodynamics and physics [1]. However, because the air flow and fluid–solid coupling is complex, it is difficult to develop an accurate physical model for the falling motion of leaves. In computer graphics, to simulate falling leaves, researchers usually use simplified physical models or leaf path templates designed by artists. Most of these methods are unsuitable for GPU parallel processing. Large quantities of falling leaves cannot be simulated by the CPU in real time. Real-time simulation of large numbers of falling leaves is still a current challenge.

In this paper, we propose a GPU based real-time approach to simulate the motion of many falling leaves. Our major contributions are:

- a data-driven approach for leaf motion synthesis, which uses six primitive motions as low-dimensional features to describe a complex falling trajectory;
- a GPU based framework for real-time simulation of large numbers of falling leaves.

The rest of this paper is organized as follows. In Section 2, we present related research and underlying technologies related to our proposed method. Our method is then outlined in Section 3. In Section 4, we describe how we use low-dimensional features to represent falling trajectories using motion synthesis. The GPU simulation framework is presented in Section 5. Experimental results are given in Section 6, followed by conclusions in Section 7.

2 Related work

Scientists began to study how to simulate the movement of light yet rigid bodies such as falling leaves as early as 1870 [2]. However, after one and a half centuries, this problem has not yet been fully solved. Prior work for the modelling of falling leaves can be divided into three categories, based on using physical models, trajectory templates, or motion synthesis.

In 1994, Tanabe and Kaneko [3] built a physical model to simulate paper falling in air. They developed a simple phenomenological model of falling paper by solving ordinary differential

1 Department of Computer Science, Zhejiang University, Hangzhou 310027, China. E-mail: C. Li, licy_cs@zju.edu.cn; J. Qian, ohday@zju.edu.cn; R. Tong, trf@zju.edu.cn (✉).

2 National Centre for Computer Animation, Bournemouth University, Poole, BH12 5BB, UK. E-mail: J. Chang, jchang@bournemouth.ac.uk; J. Zhang, jzhang@bournemouth.ac.uk.

Manuscript received: 2015-09-23; accepted: 2015-10-20

equations, based on the Kutta–Joukowski theorem. However, they failed to consider the influence of vortices, which makes their simulation less than realistic. Wei et al. [4] proposed an approach to simulate the falling motion of light-weight rigid bodies via simulation of vortices and air motions in space. The authenticity of their work has been widely acknowledged; however, due to the high computational costs, their method cannot be applied to real-time rendering.

In 2008, Vázquez and Balsa [5] used Maya to design templates for the paths of falling leaves. Before rendering, a template is chosen for the path of each leaf. This approach allowed the use of a GPU, making it possible to simultaneously simulate very many falling leaves. Li et al. [6] proposed an approach to generate a 2D trajectory for a falling leaf using simple interaction, which is applicable to 2D animation. Compared to the approach based on vortex simulation, these methods are more suited for use on a GPU, but the authenticity of the scene is compromised.

In 1997, Field et al. [7] suggested use of a Re–I^* phase diagram for free-fall motions based on statistics. They pointed out that in fluids with varying Reynolds number (Re) and dimensionless moment of inertia (I^*), various primitive motions exist, such as steady descent (SD), periodic tumbling (PT), transitional chaotic (TC) behaviour, and periodic fluttering (PF). A complex leaf falling motion can be seen as a combination of these primitive motions. After many experiments, Zhong et al. [8] found two other primitive motions, transitional helical (TH) motion and periodic spiral (PS) motion. These ideas have enabled data-driven motion synthesis to become a new way to simulate falling leaves.

In 2014, Xie and Miyata [1] synthesized a complex falling leaf trajectory by the use of motion synthesis, and used the trajectory to model falling leaves. Since each leaf has its own particular trajectory, their simulation is realistic. Compared to a simulation using a physical model, their simulation is much faster: it takes just 3 ms to update each frame when there is only one leaf in the scene. However, the leaves' trajectories need a large amount of memory to store, making this approach impractical for GPU acceleration when there are large numbers of leaves. As the number of leaves increases in the scene, the frame rendering rate rapidly decreases.

In this paper, we improve upon Xie and Miyata's work, using low-dimensional features to describe falling leaf trajectories, thus permitting GPU acceleration. In this way, we can simulate falling leaves realistically and efficiently.

3 Overview

Figure 1 illustrates our method. It has two main steps: motion synthesis, and GPU calculation. In the motion synthesis phase, we establish a trajectory dataset which builds upon collected experimental data to describe six types of primitive motions, which form the basis for the synthesis of sections of each complex falling trajectory. This allows use of low-dimensional features to describe this falling trajectory. The GPU calculation phase commences by sending the primitive-motion trajectory dataset and the leaves' trajectory features to the GPU. During run-time, we input wind field information, the system time, and leaf geometry parameters frame by frame, allowing computation of the positions of the leaves, and the reconstruction of the leaf geometry by the GPU. Finally rendering is performed.

In the rest of this paper, we focus on two problems: how to synthesize the falling motions of leaves by expressing a leaf-fall trajectory in terms of low-dimensional features, and how to simulate falling leaves on the GPU.

4 Falling motion synthesis

4.1 Primitive motions

Falling motions can be divided into several primitive motions if there is no disturbance [1, 8, 9]. A complex falling trajectory can be represented as

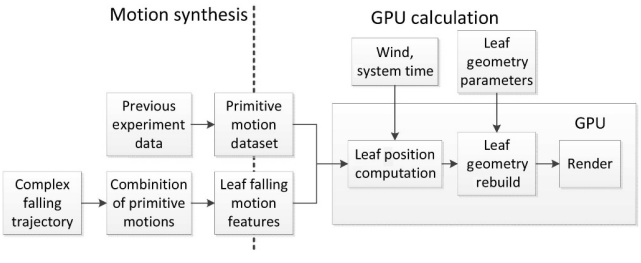


Fig. 1 Flow of our approach.

the combination of one or more primitive motion trajectories. Following Zhong et al. [8], we divide leaf falling motions [9] into 6 classes. As shown in Fig. 2, there are steady descent (SD), periodic tumbling (PT), transitional chaotic (TC), periodic fluttering (PF), transitional helical (TH), and periodic spiral (PS).

Considering these motions, SD can be seen as a uniform linear motion with slight disturbance. PT, PF, and TC are 2-dimensional motions which can be regarded as the combinations of similar trajectory segments. Following Andersen et al. [10], we may formulate the trajectory segments as

$$\begin{cases} x_t = x_0 - \dfrac{A_x}{\Omega}\sin(\Omega t) \\ y_t = y_0 - Ut - \dfrac{A_y}{2\Omega}\cos(2\Omega t) \end{cases} \tag{1}$$

where A_x and A_y are the amplitudes of vertical and horizontal velocities of the falling leaf generated by oscillations due to the surrounding viscous flow respectively, Ω describes the angular frequency of the falling motion, and U is the average descent velocity. We select segments from the function curve with $t \in \left[0, \dfrac{2k+1}{2\Omega}\pi\right]$ and $t \in \left[\dfrac{2k+1}{2\Omega}\pi, \dfrac{k+1}{\Omega}\pi\right]$. We adjust the parameters in Eq. (1) to get a trajectory segment dataset with differently shaped segments as shown in Fig. 3.

We choose suitable trajectory segments from the dataset to construct primitive motions PF, PT, and TC: for PT, we select segments with similar lengths fluttering in the same direction. For PF, we select segments with similar lengths but fluttering in opposite directions. For TC, we randomly select trajectories.

When projected onto the $x-y$ plane, TH and PS

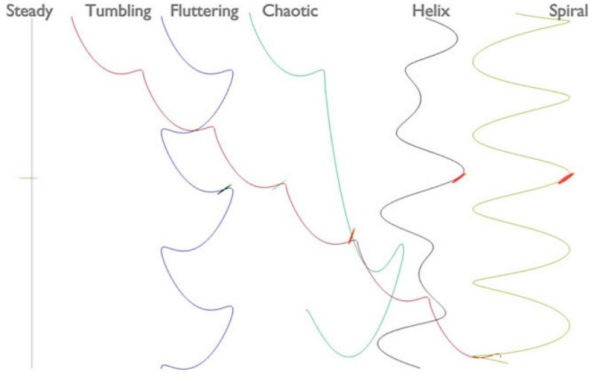


Fig. 2 Primitive motions of falling leaves.

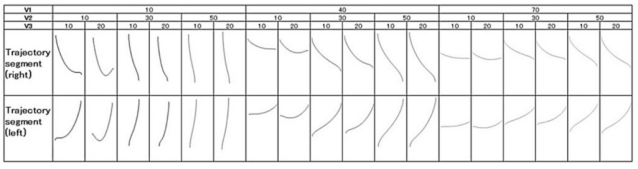


Fig. 3 Trajectory segments are obtained by varying parameters in Eq. (1). V1, V2, and V3 represent A_x, A_y, and U respectively. Ω is set to the constant value of 9.8.

curves have characteristic shapes as shown in Fig. 4. The spiral-motion trajectory is a circle while the helical-motion trajectory is similar to an eight-petal rose curve. These curves can be represented by the following equations:

$$\begin{cases} x_t = A_\mathrm{e}\cos(\Omega t)(1 + E_\mathrm{e}\sin(k\Omega t)) \\ y_t = h - Ut \\ z_t = A_\mathrm{e}\sin(\Omega t)(1 + E_\mathrm{e}\sin(k\Omega t)) \end{cases} \tag{2}$$

where A_e is the amplitude of the elliptical oscillation generated in the $x-y$ plane, E_e is the ratio of the minor axis to the major axis of the oscillation ellipse, k is the ratio of the period of elliptical oscillation to that of rotation of the falling object, and h is the height from which the object is released. When E_e is close to 0 and $k = 1$, the curve describes PS motion; when E_e is close to 1 and $k = 4$, the curve can be used to describe TH motion. Thus, we can obtain paths for different primitive motions by adjusting the parameters in the equations above.

The rotational motion while falling largely affects the realism of the leaf falling motion. Following Tanabe and Kaneko's work [3], we have the following ordinary differential equation (ODE):

$$\frac{\mathrm{d}w}{\mathrm{d}t} = -k_\mathrm{a}w - \frac{3\pi\rho V^2}{l}\cos(\alpha+\theta)\sin(\alpha+\theta) \tag{3}$$

where w is the angular velocity of the leaf, ρ is the density of the leaf, and θ and α are the angles the leaf makes with the $x-y$ and $x-z$ planes respectively, k_a is the friction against the motion perpendicular to the paper, V is the velocity in the $x-y$ plane.

Using this ODE gives authentic angular velocity

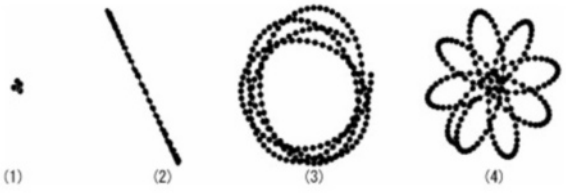


Fig. 4 Motions projected into the $x-y$ plane: (1) steady descent; (2) fluttering, chaotic, and tumbling motions; (3) spiral motion; (4) helical motion.

data for the leaf falling motion. Combining a suitable velocity and angular velocity gives a primitive-motion trajectory for a falling leaf. These are gathered into a dataset. Every primitive-motion trajectory in the dataset contains N vectors of the form (x_t, y_t, z_t, w_t), where x_t, y_t, and z_t refer to the velocity of the leaf in the x, y, and z directions at time t respectively and w_t denotes the angular velocity of the leaf at time t.

The whole leaf falling motion can be seen as the combination of many primitive motions, while the switches between these primitive motions are not arbitrary.

Xie and Miyata [1] proposed a hypothesis that the vortexes are gradually generated behind the object because of the vorticity of the surrounding flow when an object starts falling from a release point. As a consequence, the motion of a leaf becomes increasingly sensitive to internal forces, which makes the motion unpredictable.

We define the labels L_i, $i = 1, \ldots, 6$ to denote primitive motions of types SD, PT, TC, PF, TH, PS respectively. From L_1 to L_6, the randomness of the primitive motion gradually increases. A falling leaf motion M comprises a sequence of primitive motions $M = m_1 \parallel m_2 \parallel \cdots \parallel m_i$, each of which is labelled as above: $m_i = L_j$. As shown in Fig. 5, the label subscript sequence $L_1, \ldots, L_i$ should be an increasing sequence.

4.2 Low-dimensional trajectory feature of falling leaves

Xie and Miyata's approach [1] needs to precompute the whole trajectory of each falling leaf. It performs motion synthesis using the leaves' trajectories, providing strong realism but requiring large amounts of memory, making it inappropriate for simulating many falling leaves. For the same reason, it is unsuitable for GPU use. To solve the problem, we use low-dimensional features to represent leaf trajectories.

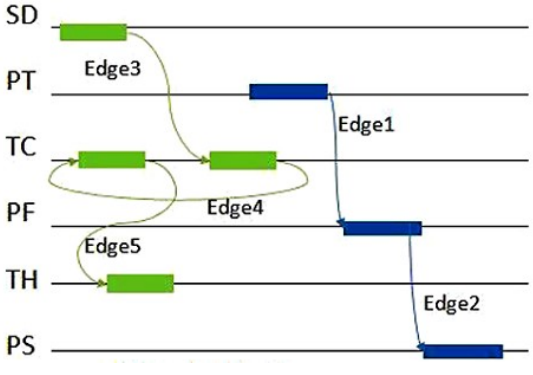


Fig. 5 Falling leaf motion examples. For the green example, $M_1 = \{L_1, L_3, L_3, L_5\}$; for the blue example, $M_2 = \{L_2, L_4, L_6\}$.

We define the feature D to be the trajectory feature for leaf falling motion, $D = \{S_1, S_2, S_3, S_4, S_5\}$, where $S_i = \{L_i, T_i, P_i\}$ denotes the current primitive motion stage, in which L_i is the primitive motion index, T_i denotes the total time for this motion stage, and P_i denotes the starting point for the primitive motion. There are five stages in D, which means there are at most 4 switches in the whole falling motion sequence. From the results of experiments in previous studies, we find that 4 switches are typically sufficient. We get L_i from Table 1, which is calculated by a Markov chain model. T_i and P_i can be obtained from the following equations:

$$\begin{cases} 0 \leqslant P_i \leqslant 1 \\ 0 \leqslant T_i \leqslant 1 \\ T_1 + T_2 + T_3 + T_4 + T_5 = 1 \end{cases} \tag{4}$$

Using the 15-dimensional feature D, we can reconstruct the whole falling-motion trajectory. Moreover, by using feature D and the primitive-motion dataset, we do not need to save the whole trajectory data into video memory. We only need to calculate the current velocity and angular velocity of the leaf.

5 GPU simulation framework

In traditional game engines, particle systems usually operate on the CPU. This places a heavy workload on the CPU, and data exchange between memory and video memory becomes a bottleneck in the simulation of complex particle motions. Recently, GPU based particle systems have been introduced, though most of them only support simple trajectory motions, and are unable to realize complex

Table 1 Motion state transition probabilities

P	L_1	L_2	L_3	L_4	L_5	L_6
L_1	0.1538	0.3462	0.0385	0.0385	0.3077	0.1154
L_2	0	0.3261	0.0217	0.0870	0.3043	0.2609
L_3	0	0	0.4444	0.0833	0.2500	0.2222
L_4	0	0	0	0.6957	0.0870	0.2174
L_5	0	0	0	0	0.9231	0.0769
L_6	0	0	0	0	0	1.0000

calculations needed for tasks such as simulating falling leaves.

To solve the problem, we have built a simulation framework for falling leaves based on the GPU—both calculations and geometry reconstruction are realized on the GPU, which reduces the workload on the CPU as well as data exchange.

As shown in Fig. 6, simulation on the GPU includes four steps: calculation of trajectories, responses of motions to wind, geometrical reconstruction, and rendering of leaves. We discuss these steps in turn in this section.

In the initial stage, we transmit the primitive motion dataset, leaf trajectory features, and initial leaf locations to the video memory. During real-time simulation, the system time and wind field information are updated at each frame. Following previous works on GPU particle systems [11–13], we define the following data streams in the compute shader:

- Geometric feature streams: these contain the geometric features of each leaf, including leaf size and normal direction, used for the geometry reconstruction, and which can be calculated on CPU.
- Trajectory feature streams: these contain the 15-dimensional features for each leaf as discribed in Section 4.2. The stream is used for free-fall motion simulation.
- Location and velocity streams: these two streams contain the location, velocity, and angular velocity of each leaf particle.

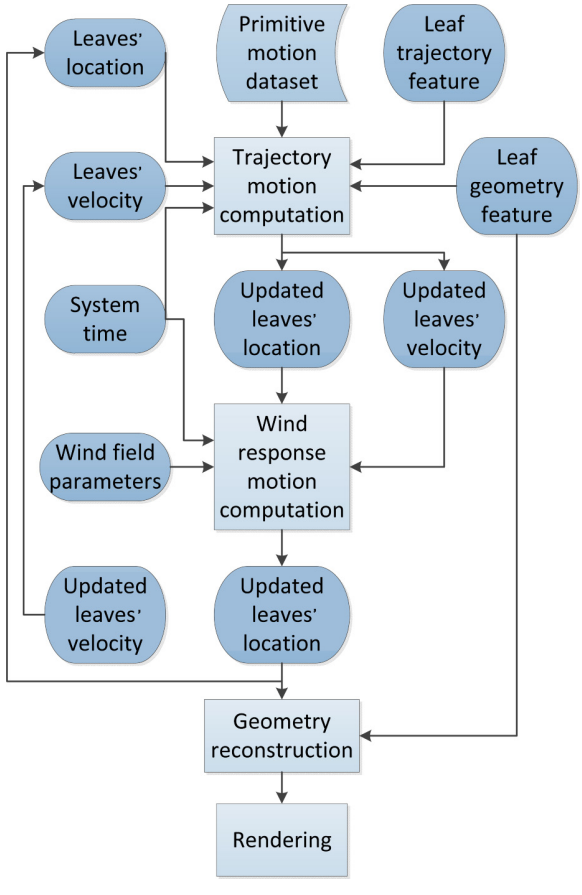


Fig. 6 GPU simulation pipeline.

In the compute shader, we first compute the leaf's position in the trajectory. Given a leaf falling motion feature $D = \{S_1, S_2, S_3, S_4, S_5\}$ and the current system time t, we get the current motion state $S_i = \{L_i, T_i, P_i\}$. Using $\odot$ to denote the sampling operation, so $L \odot i$ means primitive motion samples at location i, we get the velocity and angular velocity via the following equation:

$$V_{\mathrm{T}}, \Omega_{\mathrm{T}} = L_i \odot \left(\frac{t - t_{\mathrm{s}}}{T_i} + P_i \right) \tag{5}$$

where V_{T} and Ω_{T} are the velocity and angular velocity of the leaf's motion respectively, and t_{s} is the time when stage S_i started. In this way, we get V_{T} and Ω_{T} without reconstructing the trajectory, saving much video memory.

The simulation of wind has been discussed in many papers [14–16]. Following Mann [14], we simulate wind motion using the following equation:

$$\begin{aligned} V_{\mathrm{W}} = {} & 0.5 \\ & + 0.6 F_{\mathrm{W}} \cos(2.4 + 0.2t + x) \sin(0.4 + 0.35t + x) \\ & + 0.7 F_{\mathrm{W}} \cos(0.3 + 0.1t + x) \sin(1.2 + 0.35x) \end{aligned} \tag{6}$$

where V_{W} is the current wind speed, x is the x-coordinate in the wind field, and F_{W} is the wind strength controlled by the user. When $F_{\mathrm{W}} = 1$, the variation of wind speed is shown in Fig. 7. For each leaf, we have a parameter W_{I} which describes the wind's effects on the leaf. Equation (7) gives the leaf velocity V and angular velocity Ω in the wind field.

$$\begin{cases} V = V_{\mathrm{T}} + V_{\mathrm{W}} W_{\mathrm{I}} \\ \Omega = \Omega_{\mathrm{T}} \end{cases} \tag{7}$$

With V and Ω, we can update the leaf's position and normal information stored in the location and velocity streams.

After the computation, we have to reconstruct geometrical shape of each leaf to obtain the vertex and topology information [17]. As shown in Fig. 8, a rectangle is used to represent the leaf during

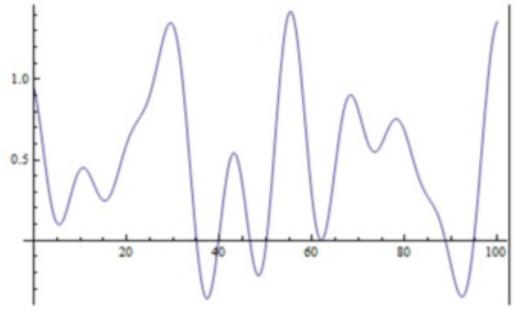


Fig. 7 Wind speed when $F_W = 1$ and $x = 0$.

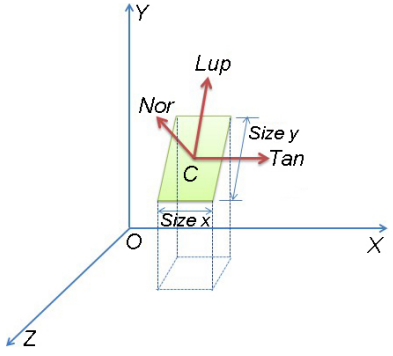


Fig. 8 Leaf rebuilding in GPU.

simulation. The representation includes C, the center of the leaf, Nor, the normal of the leaf, Tan, the tangent to the leaf, and $Size\ x$ and $Size\ y$, the sizes of the leaf. We get the Lup vector by finding the vector perpendicular to T in the plane of the rectangle, allowing us to calculate the positions of the vertices of the rectangle.

Reconstruction runs on the geometry stream in the shader. Using geometric reconstruction technology [18, 19], we can avoid transmitting vertex and mesh information from memory to video memory every frame, which greatly increases the rendering rate.

Using the geometric information for the leaves, we render them using vertex and pixel shaders in DirectX 11.

6 Results

As shown in Fig. 9, we have applied our approach in a virtual landscape application, which has substantially improved the realism of the autumn scene. In order to prove the effectiveness and the advantage of our approach to simulate large numbers of leaves, we have designed a case to compare our approach with Xie and Miyata's. We use a simple

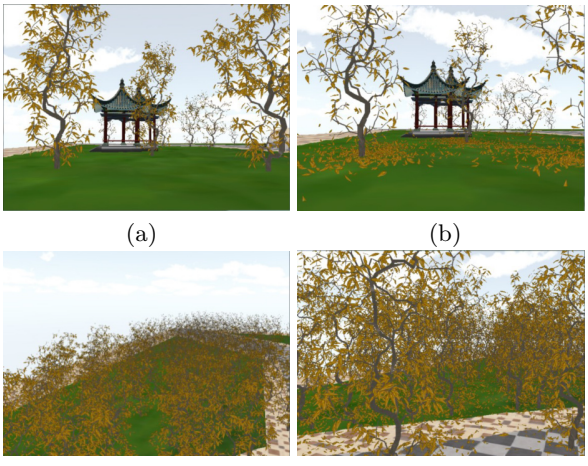


Fig. 9 Test scenes: (a) and (b) show a garden scene before and during defoliation respectively; (c) and (d) show scenes with many trees and leaves.

garden scene, shown in Figs. 9(c) and 9(d), with varying numbers of trees in it, rendering the scene with two different approaches:

1. Xie and Miyata's approach, computing a trajectory for each leaf, and updating the leaf position on the CPU;
2. our approach, using a low-dimensional feature to rebuild each leaf's trajectory, and the geometry of the leaf, on the GPU.

For comparison, both simulation approaches were built with Microsoft Visual Studio 2010 and DirectX 11. Evaluation was performed on a desktop computer running Windows 7, an Intel Core i5-4570 CPU, 8 GB of memory, and an nVidia GTX 650TI video card.

Without falling leaves, the frame rates for both examples was close to 1200 fps. We then started the simulation, ran the programs for 40 s, and recorded the average fps and CPU utilization.

Table 2 gives our test results. Approach 1 provides fewer fps and has higher CPU utilization. As the quantity of the leaves reaches 50,000, fewer than 30 fps are achieved, which is unsuitable for real-time rendering. For approach 2, even for 1 million leaves, the frame rate is still 75 fps, and the CPU utilization rate is also relatively low. Our approach clearly has

Table 2 Experimental results. Each entry gives frames per second (fps) and percentage of CPU utilization

Leaves	1000	10,000	50,000	100,000	1,000,000
Approach 1	124/17%	65/21%	14/45%	3/63%	0.05/89%
Approach 2	450/13%	350/15%	220/15%	164/15%	75/15%

a signicant advantage when simulating many falling leaves.

7 Conclusions

This study proposes an approach to realize real-time simulation of huge numbers of falling leaves based on the GPU. While ensuring realism of simulation, we have substantially improved the simulation efficiency with the ability to simulate over 1,000,000 leaves simultaneously.

This approach is not limited to the simulation of falling leaves, but is also applicable to other light rigid bodies falling such as feathers, paper scraps, and little plates.

In our future work, we would like to improve the level of realism. We intend to add a collision response mechanism to calculate collision responses between the leaves, as well as between the leaves and other objects. Such an implementation needs to be highly efficient since we will have to conduct collision detection and collision response calculations for large numbers of objects in real time.

Acknowledgements

The work is supported by National High-tech Research and Development Program of China (No. 2013AA013903).

References

[1] Xie, H.; Miyata, K. Real-time simulation of lightweight rigid bodies. *The Visual Computer* Vol. 30, No. 1, 81–92, 2014.

[2] Maxwell, J. C. On a particular case of the descent of a heavy body in a resisting medium. *The Cambridge and Dublin Mathematical Journal* Vol. 9, 145–148, 1854.

[3] Tanabe, Y.; Kaneko, K. Behavior of a falling paper. *Physical Review Letters* Vol. 73, No. 10, 1372, 1994.

[4] Wei, X.; Zhao, Y.; Fan, Z.; Li, W.; Yoakum-Stover, S.; Kaufman, A. Blowing in the wind. In: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, 75–85, 2003.

[5] Vázquez, P.-P.; Balsa, M. Rendering falling leaves on graphics hardware. *Journal of Virtual Reality and Broadcasting* Vol. 5, Article No. 2, 2008.

[6] Li, H.; Sun, Q.; Zhang, H.; Zhang, Q. Example-based motion generation of falling leaf. In: Proceedings of 2010 International Conference on Computer Design and Applications, Vol. 5, 217–221, 2010.

[7] Field, S. B.; Klaus, M.; Moore, M. G.; Nori, F. Chaotic dynamics of falling disks. *Nature* Vol. 388, No. 6639, 252–254, 1997.

[8] Zhong, H.; Chen, S.; Lee, C. Experimental study of freely falling thin disks: Transition from planar zigzag to spiral. *Physics of Fluids* Vol. 23, No. 1, 011702, 2011.

[9] Kruger, J.; Kipfer, P.; Konclratieva, P.; Westermann, R. A particle system for interactive visualization of 3D flows. *IEEE Transactions on Visualization and Computer Graphics* Vol. 11, No. 6, 744–756, 2005.

[10] Andersen, A.; Pesavento, U.; Wang, Z. J. Unsteady aerodynamics of uttering and tumbling plates. *Journal of Fluid Mechanics* Vol. 541, 65–90, 2005.

[11] Kipfer, P.; Segal, M.; Westermann, R. UberFlow: A GPU-based particle engine. In: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware, 115–122, 2004.

[12] Kondratieva, P.; Kruger, J.; Westerman, R. The application of GPU particle tracing to diffusion tensor field visualization. In: Proceedings of IEEE Visualization, 73–78, 2005.

[13] Hérault, A.; Bilotta, G.; Dalrymple, R. A. SPH on GPU with CUDA. *Journal of Hydraulic Research* Vol. 48, No. S1, 74–79, 2010.

[14] Mann, J. Wind field simulation. *Probabilistic Engineering Mechanics* Vol. 13, No. 4, 269–282, 1998.

[15] Veers, P. S. Three-dimensional wind simulation. Technical report. Sandia National Laboratories, Albuquerque, NM, USA, 1988.

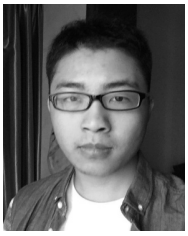
[16] Karki, R.; Hu, P. Wind power simulation model for reliability evaluation. In: Proceedings of Canadian Conference on Electrical and Computer Engineering, 541–544, 2005.

[17] Tatarchuk, N.; Shopf, J.; DeCoro, C. Real-time isosurface extraction using the GPU programmable geometry pipeline. In: Proceedings of ACM SIGGRAPH 2007 Courses, 122–137, 2007.

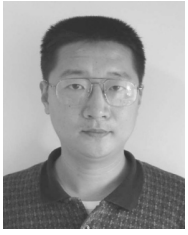
[18] DeCoro, C.; Tatarchuk, N. Real-time mesh simplification using the GPU. In: Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games, 161–166, 2007.

[19] Lorenz, H.; Döllner, J. Dynamic mesh refinement on GPU using geometry shaders. In: Proceedings of the 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, 2008.

Chengyang Li received his B.S. degree in biomedical engineering from Zhejiang University. He is a master degree candidate in the Department of Computer Science and Engineering, Zhejiang University, Hangzhou, China. His research interests include computer graphics, visual media computing, and computer vision.



Jingye Qian is a M.S. degree candidate in the Department of Computer Science, Zhejiang University, China. He received his B.S. degree from Nanjing University of Aeronautics and Astronautics, China, in 2012. His research interests include image processing and computer graphics.



Ruofeng Tong received his B.S. degree in mathematics from Fudan University, and Ph.D. degree in applied mathematics in 1996 from Zhejiang University. He continued his research as a postdoctoral researcher at Intelligent Systems and Modeling Laboratory, Hiroshima University, Japan. Currently, he is a professor of the Department of Computer Science and Engineering, Zhejiang University, Hangzhou, China. His research interests include computer graphics and CAD, medical image reconstruction, virtual reality.

Jian Chang received his Ph.D. degree in computer graphics at the National Centre for Computer Animation (NCCA), Bournemouth University, UK, in 2007. He is now an associate professor at the NCCA and a member of the Computer Animation Research Centre. His research has focused on a number of topics related to deformation and physically-based animation, geometric, algorithmic art, character rigging and skinning. He has conducted research in applications related to medical simulation and virtual surgery, involving complex modeling work of human anatomic structures and dynamics.

Jianjun Zhang received his Ph.D. degree from Chongqing University in 1987. He is a professor of computer graphics at the National Centre for Computer Animation, Bournemouth University, UK, and leads the Computer Animation Research Centre. He is also a cofounder of the UK's Centre for Digital Entertainment, funded by the Engineering and Physical Sciences Research Council. His research focuses on a number of topics related to 3D virtual human modelling, animation and simulation, including geometric modelling, rigging and skinning, motion synthesis, deformation and physics-based simulation.