An Optimization Method for Real-Time Natural Phenomena Simulation on WinCE Platform

Ning Liu
School of Software, Sun Yat-Sen University, 510275 P.R. China

Rong Li
School of Information science and technology, Sun Yat-Sen University, 510275 P.R. China

Zhenyu Yang
School of Information science and technology, Sun Yat-Sen University, 510275 P.R. China

Hongyang Chao*
School of Software, Sun Yat-Sen University, 510275 P.R. China
Corresponding author
E-mail: isschy@mail.sysu.edu.cn

Abstract

We present a new optimized algorithm of particle system for simulating flame, so that it can be implemented on WinCE platform which has only limited resources. Our approach mainly intends to store some traces first and then load them randomly during the simulation to reduce the computation complexity of the particle system. We have tested this approach in the simulation of a flame, and our method shows a significant performance gain with little loss in the visual appearance of the simulation, this indicates the potential to extend our approach to particle simulation involved with other natural phenomena for resource limited platforms.

1. Introduction

The natural phenomena simulation, which has broad applications in game scenes, is an attempt to study how to evoke common, natural phenomena in the computer such as flame, rain, snow, fog, clouds, water, trees and so on. Due to the rich hardware resource of PC platforms, the PC-based simulations of natural phenomena usually can get good visual effects and have many applications. However, since WinCE or other embedded platforms only have limited hardware resource; it is difficult to simulate some natural phenomena realistically, which greatly impedes the development of mobile game industries.

After years of research, there have been many methods for PC-Based simulations of natural phenomena. Mandelbrot [1] proposed a fractal method for simulating natural phenomena that have fractal characteristics. Aristid Lindenmayer [2] proposed a method called of L.Grammar System for studying the morphology and the growth of a plant. Reeves [3] proposed using a particle system model to simulate objects with fuzzy borders.

This paper will mainly discuss a flame simulation on a Windows CE platform using a particle system model. Due to the relatively large number of particles in a particle system, the computation in real-time necessary for calculating each particle will be complex and time-consuming. Furthermore, the real-time effects will not be realistic. Therefore, it is necessary to optimize the performance of the particle system. Addressing the unique characteristics of the WinCE platform such as weak computation abilities, this paper presents a method for flame simulation by randomly retrieving the traces of particles saved previously.

In this paper, the content will be organized as follows: Section II gives an introduction to the particle system used for flame simulation. Section III describes our method for optimization. Section IV gives the experimental results in detail. Section V is a summary and outlook for the further research.

2. Particle System for Flame Simulation

2.1. Particle system

So far the particle system is one of the most successful methods for simulating irregular and fuzzy objects since it was proposed by Reeves [3] in 1983. This method represents some fuzzy objects as a cloud of primitive particles with attributes. Changes in the
particle attributes will change the attributes of the simulated object. The major steps of rendering a scene by using a particle system can be expressed in the following flowchart:

![Flowchart of Particle System Processing](image)

**Figure 1 Processing of Particle System**

### 2.2. Particle system for flame simulation

#### 2.2.1. Attributes of particles

To simulate natural phenomena realistically with a particle system, we should make sure that the attributes of each particle in the particle system are as similar as possible to the attributes of the counterpart in the natural phenomena. As to flame simulation, the major attributes that particles should have can be seen in the following picture:

![Flame Particle Attributes](image)

**Figure 2 The property of a flame particle**

#### 2.2.2. Change of particle attributes

From the above picture we find that, the flame particles are generated at the flame root, upwardly raised, and extinguished at the end. The particles at different places have different attributes. Therefore, we have to change the attributes of each particle as they move upwardly. As a particle is produced into the system, it is assigned its initial attributes. In the running process, the dead particles which have past their prescribed lifetime are removed from the system, and at the same time some new particles are added in so that the number of particles remains relatively stable. The particles remaining in the particle system are moved and changed according to their dynamic attributes. For example, a particle's color changes over time as prescribed by the rate-of-color-change parameter. The transparency and size of particles are changed in exactly the same way.

### 3. Optimization

#### 3.1. Prior optimization methods

Using fixed-point algorithm instead of floating-point algorithm can improve the performance [4]. However, fixed-point operations may lose accuracy, and the improvement of performance is limited. For some complex trigonometric calculations, it is not easy to simulate such operations by using fixed-point algorithm. Block processing method was proposed by David O'Brien[5]. Time-sharing dealing method was proposed by Jim X. Chen[6]. Mu-Yuan, Wang [7] proposed in his paper to simplify the particle system by reducing the particles’ unimportant attributes.

#### 3.2. Our optimization method

Traditional simulating methods that use particle system to represent natural phenomena mainly attempt to calculate the particles’ attributes on each frame, so we must calculate all the particles’ attributes for the attribute update.

The attributes of a particle include the particle’s color, running-speed, position, etc. In the update process, multiplication is mainly involved in the calculation.

In order to reduce this calculation, we store several particles’ movement traces in the trace-storeroom which will be used later.

As shown in Fig 3, discrete nodes are one particle’s positions in the whole life cycle, and each position corresponds to the relevant attributes which are stored in the trace-storeroom. When drawing, we just load a particle’s trace from the trace-storeroom randomly. Actually, we make use of the difference between store/load time and calculation time to achieve an
Step 1: Calculation of particle trace

At first, let us assume that we need the traces of \( n \) particles and every particle's trace comprises 10 discrete nodes which are evenly distributed in the particle's life cycle. A particle's life cycle can be expressed in the following formula: \( Life = DeadTime - StartTime \), where \( DeadTime \) is the time the particle dies and \( StartTime \) is the time the particle is created.

So, when a particle reaches the \( i^{th} \) node, its life value is just \( SubLife \) which refers to the life time of a particle. This can be expressed in the formula of \( SubLife = Life/i \times 10 \). We then calculate the particle's attributes according to this life value. Therefore, after one life cycle we get 10 discrete nodes, and the attributes of these 10 nodes are stored in memory or database.

After the first round of calculation, we get \( n \) traces, and these traces are stored in one dimensional array. The total amount of data storage is:

\[
s = k \times n \times \sum_{i=1}^{n} \text{sizeof}(Attribute_i)
\]

(1)

Where \( k \) is the number of nodes for each trace. For instance, if we assume \( k \) equals 10, \( n \) is the number of traces we have stored. \( Attribute_i \) is the particle's \( i^{th} \) attribute and \( m \) is the amount of a particle's attributes.

Step 2: Rendering particles

After we store all of the traces we want, the second step is the particle's rendering. Let us assume we need \( m \) (\( m >> n \)) particles to draw current frame. Now we can load the attributes of traces nodes stored before to replace the calculation of particle attributes.

For example, once particle \( j \) is created, we choose a trace from trace-storeroom randomly, and copy the first node's attributes to \( j \). And every time an update is needed, the particle \( j \)'s attributes are just the attributes of next node. That is, there are 10 different nodes in rendering particle \( j \), and each new node’s attributes come from the corresponding nodes of the loaded trace. When particle \( j \) runs to the tenth node, it is ready to die. Likewise, when it is necessary to create another particle \( k \), we will also get a trace randomly in the same way.

3.3. Optimization Efficiency Analysis

As shown above, a particle's attributes include size, alpha, color, velocity, position and acceleration. Here size, alpha are one dimensional float variables. Color, velocity, position are three dimensional float variables. Acceleration is a constant number in this paper, which can be ignored in the sense of resource consumption.

As all we know, the computer's store/load data time is only about 1/4 time of multiplication/division, and it is almost equivalent to that of addition or subtraction. Assume that the time to store/load a float is equal to 1 unit, so the total amount of time to load/store a particle’s size, alpha, color, velocity and position, will consume 11 time units. However, if we need to calculate those values, we have to follow the formula below:

\[
Size = Size + SChange \times \Delta T
\]

\[
Alpha = Alpha + AlChange \times \Delta T
\]

\[
Color = Color + CChange \times \Delta T
\]

\[
Velocity = Velocity + Acc \times \Delta T
\]

\[
Position = Position + Velocity \times \Delta T
\]

(2)

(3)

(4)

(5)

(6)

Here \( SChange \) is the change of a particle speed value between two adjacent frames. \( AlChange \) is the change of particle alpha value, and \( Acc \) is particle acceleration, and \( \Delta T \) is the time interval between two frames. As mentioned above, the computing cost for multiplication or division is 4 time unit, and the cost for addition or subtraction is 1. So the cost for each update of the particle is equal to \( 3 \times 3 \times (1+4) + 2 \times (1+4) = 55 \) time unit. The consumption time for the two methods is contrasted as follow:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Cost</th>
<th>Store/Load</th>
<th>Calculation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>1</td>
<td>1+4</td>
<td></td>
</tr>
<tr>
<td>Alpha</td>
<td>1</td>
<td>1+4</td>
<td></td>
</tr>
<tr>
<td>Color</td>
<td>3+1</td>
<td>3*(1+4)</td>
<td></td>
</tr>
<tr>
<td>Velocity</td>
<td>3+1</td>
<td>3*(1+4)</td>
<td></td>
</tr>
<tr>
<td>Position</td>
<td>3+1</td>
<td>3*(1+4)</td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>11</td>
<td>55</td>
<td></td>
</tr>
</tbody>
</table>

This shows, after using our optimization method, the consumption time was reduced by a value of 44 during every particle update once. Therefore assume we have \( m \) particles, every particle has 10 nodes, after every particle has finished the entire life cycle drawing, the run time units savings equal to \( 44 \times m \times 10 \).

4. Experimental Results

We used Pocket PC 2003 and OpenGL/ES as the experiment tools. \( Fps \) (frame per second) is an important parameter to judge the simulation efficiency. We can see in Tab 2, for simulating a flame; our new algorithm which used optimization method above has higher fps than traditional methods in the same particle...
number. Here, traditional method is the one which only use calculation to get all the attributes of each particle.

<table>
<thead>
<tr>
<th>Method</th>
<th>Frame Per second</th>
<th>Particle Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Traditional method</td>
<td>23.9</td>
<td>200</td>
</tr>
<tr>
<td>Traditional method</td>
<td>16.1</td>
<td>300</td>
</tr>
<tr>
<td>Traditional method</td>
<td>13.7</td>
<td>400</td>
</tr>
<tr>
<td>Optimization method</td>
<td>32.9</td>
<td>200</td>
</tr>
<tr>
<td>Optimization method</td>
<td>21.3</td>
<td>300</td>
</tr>
<tr>
<td>Optimization method</td>
<td>17.6</td>
<td>400</td>
</tr>
</tbody>
</table>

Based on the two following figures, we can compare the visual effects between optimized and non-optimized methods. Fig 4 shows the visual effects of a traditional method. Its fps equals to 16.1. Fig 5 is the visual effects of our optimizing method and its fps equals to 21.3. Both particle systems’ parameters are the same. Traditional method consumes more time than our algorithm to build up the particle system.

This proves that our optimization method is a feasible method and it will not undermine the visual effect of phenomenon stimulation.

What’s more, referring to the traditional methods for smoke and rain simulation which had been discussed in detail by Ronald Thorsten Holtkamper[8] and Zhong-Xin Feng[9], we have used our method for simulating smoke and rain, and got a good gain in system performance with little loss of visual appearance.

5. Conclusions

In this paper we have presented a new method which is based upon randomly loading traces to reduce a particle system’s calculation, so that it becomes possible to simulate some natural phenomena realistically in real time on WinCE platforms. The new algorithm gets faster performance and gives the similar visual simulation effects with little loss. We hope our work can make a contribution to the development of mobile game engine.

But the total time consumed for simulating natural phenomena with a particle system comprises two kind times, the calculation time and rendering time. Our method is only an optimization for the calculation time. So, if we can reduce rendering time further, the efficiency of system may significantly increase.

6. References