基于软粒子的实时沙尘暴仿真

许 森[1,2]，项 予[2]

（1. 四川大学 计算机学院，四川 成都 610064；
2. 四川大学 视觉合成图形图像技术国防重点学科实验室，四川 成都 610064）

摘 要：针对沙尘暴仿真中出现的实时性和高效性不能满足用户需求等问题，提出了基于软粒子的沙尘暴仿真方法。采用可编程技术，在GPU上对粒子进行软化，在粒子和场景交互时，能够达到平滑过渡的效果，使其有更强的真实感。在视点周围产生和绘制软粒子，在顶点着色器中对软粒子的属性进行更新，能够达到实时性绘制。根据粒子运动的半径，利用线性插值方法使其能淡入淡出，有效避免了闪烁现象，最终达到实时、逼真、高效的沙尘暴仿真，大幅度提高了三维场景的渲染效率和真实感。

关键词：软粒子；沙尘暴；逼真性；实时性；高效性；着色语言

中图法分类号：TP391.9 **文献标识号**：A **文章编号**：1000-7024（2013）07-2503-04

Real-time sandstorm simulation based on soft particle

XU Sen[1,2], XIANG Yu[2]

(1. Computer College, Sichuan University, Chengdu 610064, China;
2. State Key Laboratory of Fundamental Science on Synthetic Vision, Sichuan University, Chengdu 610064, China)

Abstract: To deal with the problems that the effects of real-time and efficiency cannot meet the needs of the users appeared in the sandstorm simulation, a solution based on soft particles is put forward, which can simulate real-time sandstorm. Firstly, the particles is softened in the GPU by using programmable technology, when the soft particles intersect with the scene, the effect will be more smooth and verisimilar. Secondly, particles around the viewpoint is produced and rendered, and the attributes of particles is updated in the vertex shader, which can achieve real-time rendering. Finally, according to the radius of soft particles moving, using the linear interpolation method makes them gradually change, which avoids flashing phenomenon. In the end, real-time, verisimilar and efficient sandstorm simulation is improved.

Key words: soft particle; sandstorm; verisimilar; real-time; efficiency; shading language

0 引 言

自粒子系统仿真不规则物体运动以来，其得到了广泛的应用和研究[1-2]。文献[3-5]在传统的雨、雪、雾、沙尘暴等自然现象的仿真中，都是基于粒子系统的。在仿真的过程中，大部分工作是由CPU负责的，包括粒子的产生和位置的更新等，得到最终的粒子位置数据交由GPU绘制[6]。虽然基于粒子系统的自然现象仿真能达到较为逼真的效果，但是需要绘制成千上万个粒子，因此耗费了大量的计算机资源，效率较低。

鉴于此，本文提出一种新的基于软粒子的仿真，只需由CPU完成少数粒子的创建，然后系统使用几何着色器进行粒子由单个点到两个三角形（一个四边形）的几何扩展，在顶点和片元着色器中对其进行纹理贴图和着色。在GPU里完成沙尘暴粒子的软化及其更新，减轻了CPU的运算压力，充分利用了GPU的强大并行运算能力，从而高效的利用计算机的计算资源。

1 OpenScenceGraph，GLSL简介

1.1 OpenScenceGraph

OpenSceneGraph简称OSG，它是一个开源的场景图形管理开发库，主要为图形图像应用程序的开发提供场景管理和场景渲染优化功能，能对整个三维空间进行统一的管理。OSG具有跨平台性，可运行在Windows和大多数类型的Linux操作系统上。它在虚拟现实、3D游戏以及可视化

收稿日期：2012-09-19；**修订日期**：2013-02-20

基金项目：国家自然科学基金项目（60903118、60832011）

作者简介：许森（1987-），男，河南周口人，硕士研究生，研究方向为计算机图形学、虚拟现实；项予（1989-），男，四川自贡人，硕士研究生，研究方向为计算机图形学、虚拟现实。E-mail：xusen198704@163.com

仿真系统等领域都有着很广泛的应用[7]。

1.2 GLSL

GLSL 是着色语言的简称，图形处理器硬件技术更新日新月异，出现了可编程管线这一概念。传统的固定管线功能可以有顶点着色器、几何着色器和片元着色器所代替，提供了可编程的灵活性。使用于 GPU 可编程管线的着色语言也应运而生，在 GPU 可编程发展的早期，提出着色语言是为了加强对图形绘制算法的控制，但是随着科学技术的革新，目前的可编程着色语言已经应用于通用的科学计算研究。着色语言的发展给予了程序开发人员灵活而快捷的编程方式，并且能够最大可能的控制图形渲染的过程，同时利用图形硬件的并行性来提高图形绘制算法的效率。目前，主流的可编程着色语言有 3 种：基于 Direct3D 的高级着色语言（HLSL），基于 OpenGL 的着色语言（GLSL）和 NVIDIA 公司的 Cg 着色语言[8-10]。

2 粒子软化技术

粒子 sprite 被广泛应用于自然现象仿真和大型游戏场景中，它们可以用于仿真烟雾、沙尘、爆炸以及魔法等半透明效果。然而，当粒子 sprite 和场景交互时，通常会出现不自然的硬边缘效果。一种简单的解决方法是使用 DirectX10 的新特性读取深度缓冲作为纹理，使粒子 sprite 能淡出 3D 场景，软化不自然的硬边缘。

2.1 绘制粒子

每一个粒子 sprite 都是由几何着色器（geometry shader）创建的，图 1 是一个典型的粒子 sprite。首先，CPU 为每个粒子 sprite 提供一个位置点给 GPU，几何着色器将每个点扩展成为一个四边形。

图 1 粒子 sprite 图

由于粒子 sprite 是扁平状的，但是在 3D 空间中要表示的是一个球形的烟雾模型，所以可以采用视线追踪技术（bill boarding）[11-12]技术来实现，将几何着色器扩展的四边形（该四边形用来纹理映射，即纹理映射平面）朝向视点，使纹理映射平面与视线垂直，随着观察角度的变化，纹理映射平面方向也随之变换。图 2 所示，当视点 1 旋转到视点 2 时，纹理映射平面也随着转动，这样在 3D 场景中，一个扁平的粒子 sprite 就能实现球形的烟雾效果，从而达到了立体效果。

2.2 粒子软化原理

把粒子 sprite 和 3D 场景进行混合，以达到软化效果，其软化原理如图 3 所示。

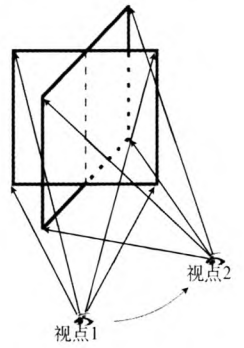


图 2 纹理映射平面随视点变化

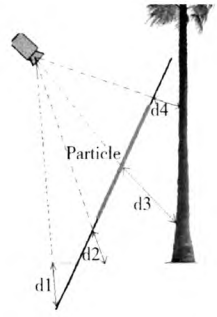


图 3 粒子软化原理

（1）粒子 sprite 距离场景很近，这样就需要将粒子 sprite 和场景进行混合，更加的淡化粒子 sprite 而显示更加清晰的场景，如图 3 中的 d4 和 d2 部分。

（2）粒子 sprite 距离场景比较远，这样场景就比较模糊，场景被粒子 sprite 覆盖，只需显示粒子 sprite 即可。如图 3 中的 d3 部分。

（3）粒子 sprite 在场景的后面，完全被场景所阻挡，这样只需丢弃粒子 sprite 而显示场景即可。如图 3 中的 d1 部分[12]。

2.3 粒子软化算法实现

首先，在片元着色器中对纹理进行采样，根据目标纹理和纹理查询坐标，通过纹理采样函数得到纹理中的信息。本文中，使用 texture2D（*uTex*，*coord*）（其中 *uTex* 是查找的目标纹理，*coord* 是查找的纹理坐标）对粒子和场景深度进行采样，片元纹理（fogColor）和场景深度纹理（sceneDepth），分别用于片元输出和深度计算。

然后，在片元着色器中进行深度计算。利用片元着色器的特殊变量 **gl _ FragCoord.** z 来获得当前粒子片元的深度值，用于和场景深度值进行比较。

当 **gl _ FragCoord.** z - **sceneDepth.** x＞0.0f 时，表示粒子片元被场景所遮挡，则丢弃（discard）粒子片元被遮挡的部分。

当 abs（**gl _ FragCoord.** z - **sceneDepth.** x）＜0.1 时，表示粒子片元接近于场景，则需要对粒子片元和场景进行混合。计算混合比例因子 scaleValue，如式（1），求得更佳比例因子，以达到更好混合效果。

当粒子片元距离场景比较远时，场景被粒子片元覆盖，则不需要将粒子片元和场景进行混合，而直接输出粒子片元即可

$$\text{scaleValue}=\text{abs}(\text{gl\_FragCoord.z}-\text{sceneDepth.x})/\text{parameter} \quad (1)$$

通过以上算法，从而能实现粒子软化效果。

2.4 效果对比

通过对粒子软化处理，仿真效果得到了明显的改善，其效果更加真实。如图 4 是在粒子软化前，可以看到在粒子和场景的交互时，有明显的不自然的硬边缘，不符合自然现象。然而，如图 5 是粒子进行了软化处理后的效果，可以看出在粒子和场景交互时，消除了不自然的硬边缘，能到达很自然的过渡效果，更加接近于自然现象。

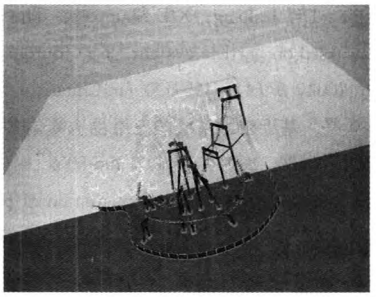


图 4　粒子软化前

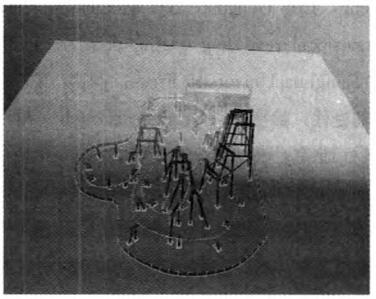


图 5　粒子软化后

3 沙尘暴实现

3.1 沙尘暴粒子的绘制

传统的沙尘暴粒子是在世界坐标中绘制的，当视点移动的过程中，粒子不随之改变，视点很快移出沙尘效果范围，这样仿真效果不够真实。本文中采取的是在视点周围绘制沙尘粒子，使得视点一直在沙尘暴中心，更加具有真实性。

粒子在 CPU 里创建，然后交给 GPU 来对粒子进行扩展，片元处理和重新绘制。首先，根据风向 **m _ WindDir** 可计算出 cos（θ），如式（2）。然后，根据式（3）可以计算出 sin（θ）。最后，根据式（4）与式（2）和式（3）的计算结果可计算出 cos（$\theta\pm\varphi$），则 R * cos（$\theta\pm\varphi$）即为粒子绘制区域，如图 6 所示

$$\cos(\theta) = -m\_WindDir \tag{2}$$

$$\sin(\theta) * \sin(\theta) + \cos(\theta) * \cos(\theta) = 1 \tag{3}$$

$$\cos(\theta\pm\varphi) = \cos(\theta) * \cos(\varphi) \mp \sin(\theta) * \sin(\varphi) \tag{4}$$

3.2 沙尘暴粒子的旋转

在几何着色器中，GPU 将 CPU 绘制的粒子扩展成为

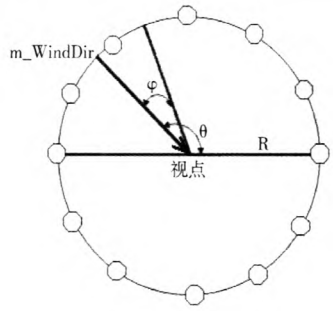


图 6　粒子绘制区域

一个四边形，为了使得效果更具有真实性，可以让四边形的每个点以相同的速度旋转。4 个点的位置随着角度的变化而实时变化，以达到沙尘暴翻滚的效果，4 个点位置的变化如式（5）所示

$$Position = t \pm r * \mathbf{sin}(angle) * rightAxis \pm r * \mathbf{cos}(angle) * upAxis \tag{5}$$

式中：t——当前粒子的位置，r——四边形对角线的二分之一，***rightAxis***——x 正半轴方向，***upAxis***——z 轴正半轴方向。

3.3 沙尘暴粒子的更新

为了保证沙尘暴不间断，需要对粒子实时更新绘制，在 CPU 里计算粒子移动距离，如式（6）所示。根据粒子移动距离，在 GPU 里对粒子顶点移动，如式（7）所示

$$Offset = prevOffset + m\_WindDir * \mathbf{m\_speed} \tag{6}$$

式中：***prevOffset***——上一帧粒子偏移量，***m _ WindDir***——风向，m _ speed 是每个粒子的移动速度

$$gl\_Position = gl\_Vertex + offset \tag{7}$$

式中：***gl _ Vertex***——当前顶点的位置，***offset***——当前点的偏移量。

当粒子移动距离（pDistance），如式（8）所示，超出半径为 R 的活动区域时，旧粒子消失，新粒子重新绘制，这样就得到了源源不断并且平滑变化的实时沙尘暴效果

$$\mathbf{pDistance} = \mathbf{sqrt}(prevOffset.x() * prevOffset.x() + prevOffset.z() * prevOffset.z()) \tag{8}$$

式中：***prevOffset***——上一帧粒子偏移量。

3.4 沙尘暴粒子的淡入淡出

为了避免沙尘暴粒子重新绘制和移出绘制区域时出现的闪烁现象，当粒子重新绘制和移出绘制区域时，可以将其和场景进行混合。首先，根据式（9）计算得混合比例因子，通过调整 parameter 可以得到不同的比例因子。然后，根据式（10）得到粒子在移动的过程中片元颜色的权重值。经过上述处理之后，粒子能平滑的进入和移出绘制区域，有效克服了闪烁现象

$$\mathbf{Value} = \mathbf{abs}((\mathbf{radius} - \mathbf{pDistance}) / \mathbf{parameter}) \tag{9}$$

式中：radius——绘制区域半径，pDistance——粒子移动距

离，parameter——变量，以求得更佳比例因子，以达到更好混合效果

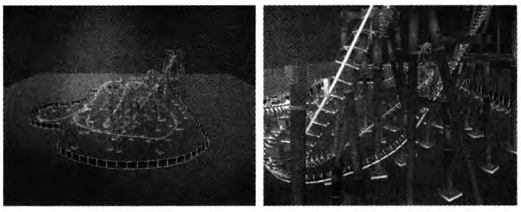
$$\textbf{fogFactor}=\textbf{clamp}(value, 0.0, 1.0) \quad (10)$$

式中：clamp 函数是将 value 值在 [0.0，1.0] 上进行截取。

4 实验结果

本文方法实现的硬件环境为：Intel（R）Core（TM）2 Duo CPU E7400 2.90GHz 处理器，4G 内存，NVIDIA GeForce GTX260 显卡。

实验程序运行图如图 7 所示。



(a) 远景图 (b) 近景图

图 7 软粒子实时沙尘暴效果

5 结束语

本文提出了一种基于软粒子的实时沙尘暴仿真方法，其创新之处在于仅仅绘制少量粒子来仿真大规模沙尘暴场景效果。同以往基于粒子系统的自然现象仿真相比，本文方法实时性好，更重要的是渲染效率得到了很大的改进。但是，本文实现的效果还有不足之处，当运动粒子重叠是会出现亮斑，粒子片元颜色没有随机改变。这些不足之处需要继续研究改进。

参考文献：

[1] LIU Xiaoling，YANG Hongyu，GUO Huqi. Real-time simulation of rain and snow in large-scale scene based on GPU particle system [J]. Computer Engineering and Design，2012，33 (6)：2399-2341 (in Chinese). [刘小玲，杨红雨，郭虎奇. 基于 GPU 粒子系统的大规模雨雪场景实时模拟 [J]. 计算机工程与设计，2012，33 (6)：2399-2341.]

[2] HU Zhenghua. Rain rendering based on programmable GPU [D]. Hangzhou：Zhejiang University，2010 (in Chinese). [胡振华，基于可编程 GPU 的雨天特绘制 [D]. 杭州：浙江大学硕士学位论文，2010.]

[3] LUO Yong，LIU Jianguo. The design of object oriented universal particle system and its implementation in fire simulation [J]. Science Technology and Engineering，2008，24 (13)：4174-4178 (in Chinese). [罗勇，刘建国. 面向对象的通用粒子系统设计及其火焰模拟中的应用 [J]. 科学技术与工程，2008，24 (13)：4174-4178.]

[4] WANG Wei，XIN Guodong. Research of real-time fire rending method based on particle system [J]. Microcomputer Information，2008，24 (13)：2-6 (in Chinese). [王巍，辛国栋. 基于粒子系统的实时火焰绘制方法研究 [J]. 微型计算机信息，2008，24 (13)：2-6.]

[5] LUO Weijia，DU Jinkang，XIE Shunping. The real-time rain simulation based on particle system [J]. Journal of Image and Graphics，2004，9 (4)：495-500 (in Chinese). [罗维佳，都金康，谢顺平. 基于粒子系统的三维场地降雨实时模拟 [J]. 中国图象图形学报，2004，9 (4)：495-500.]

[6] WU Yinxia. The design and implementation of particle effects system based on OpenGL [D]. Chengdu：University of Electronic Science and Technology of China，2009 (in Chinese). [吴银霞. 基于 OpenGL 的 3D 粒子特效系统设计与实现 [D]. 成都：电子科技大学硕士学位论文，2009.]

[7] XIAO Peng，LIU Gengdai，XU Mingliang. OpenSceneGraph three-dimensional rendering enging programming guide [M]. Beijing：Tsinghua University Press，2010：2-7 (in Chinese). [肖鹏，刘更代，徐明亮. OpenSceneGraph 三维渲染引擎编程指南 [M]. 北京：清华大学出版社，2010：2-7.]

[8] Rost RJ. OpenGL shading language [M]. Beijing：Posts & Telecommunications Press，2006：23-25 (in Chinese). [Rost RJ. OpenGL 着色语言 [M]. 北京：人民邮电出版社，2006：23-25.]

[9] OpenGL Architecture Review Board. OpenGL programming guide [M]. Beijing：Posts& Telecommunications Press，2005：11-34 (in Chinese). [OpenGL 体系结构审查委员会. OpenGL 编程指南 [M]. 北京：人民邮电出版社，2005：11-34.]

[10] LaMothe A. Tricks of the 3D game programming gurus-advanced 3D graphics and rasterization [M]. LI Xiangrui，CHEN Wu，transl. Beijing：Posts& Telecommunications Press，2005：51-62 (in Chinese). [LaMothe A. 3D 游戏编程大师技巧 [M]. 李祥瑞，陈武，译. 北京：人民邮电出版社，2005：51-62.]

[11] Mason McCuskey. Special effects game programming with DirectX [M]. KE Peng，transl. Beijing：Science Press，2006：21-25 (in Chinese). [M. 麦卡斯基. DirectX 特效游戏程序设计 [M]. 柯鹏，译. 北京：科学出版社，2006：21-25.]

[12] NVIDIA. Soft particles [EB\OL]. [2007-02-11/2012-09-25]. http：//developer. download. nvidia. com/whitepapers，2007.