交互式角色运动特效的光滑粒子流体动力学仿真

徐添辰[1] 吴恩华[1,2]

[1]（澳门大学科技学院 澳门 999078）

[2]（中国科学院软件所计算机科学国家重点实验室 北京 10019）

摘 要 长期以来，由于流体仿真和物体变形计算都具有相当程度的复杂性，使得流体与刚体的交互模拟，特别是和带有复杂动画的角色交互的效果，难以达到实时计算和渲染。在此，笔者提出了一个新的方法，用于生成沿着角色运动而产生交互的流体特效。为了实现这类效果的生成，控制流体特效与运动角色的交互，首先针对角色运动轨迹进行跟踪，根据轨迹的几何性质而生成初始状态的流体特效；然后借助光滑流体动力学（SPH）对流体粒子进行仿真。其中针对基于SPH技术的复杂性，流体仿真的过程借助GPU并行计算的能力，采用了一种新的高效粒子搜索算法，最终实现普通用户级个人计算机上实时渲染具有流体运动特征的角色运动特效。

关键词 角色动画；光滑粒子流体动力学（SPH）；流体与刚体交互；GPU搜索计算；运动跟踪

Smoothed-Particle Hydrodynamics Simulation for Interactive Motion Effects

XU Tian-chen[1] WU En-hua[1,2]

[1](*Faculty of Science and Technology, University of Macau, Macau* 999078)

[2](*State Key Lab of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing* 100190)

Abstract In the past work for long time, since the computation required on object deformation and interaction is intensive, when fluid is interacted with rigid bodies, or especially with animating figure, the demand of real-time simulation and rendering could be hardly achieved. This paper presents a novel approach for generating effects simulated by fluid dynamics and interacted by the figure motions. In order to handle the interaction between fluid effect and deformed figure, firstly, the motion trajectory of character is tracked, and then the fluid dynamics is simulated by the model of Smoothed-Particle Hydrodynamics (SPH). Moreover, during the fluid simulation, an efficient algorithm for particle searching is also proposed, in virtue of parallel processing by GPU. Consequently, the simulation of 3D fluid effects with realistic character interaction can be rendered on a consumer-level PC in real-time.

Keywords character animation; SPH; fluid – rigid body interaction; GPU computation; motion tacking

1 引 言

在众多虚拟现实与游戏应用中，人们对于计算机图形中的真实感细节和艺术效果的追求越来越趋近完美。回顾过去的系统，许多譬如水流、魔法等特效都是由传统粒子系统来实现的。其中粒子之间不存在交互作用，一些真实的细节和运动感就难以表达。为了增强特效运动的真实感，各种基于物理的仿真技术从而被引进。但与此同时，这些技术要以大量的运算作为代价，来获得逼真的结果。尤其是基于流体的仿真，由于流体的拓扑性质时时刻刻都在改变，于是相比其他物理模拟而言，具有更高的复杂度。

本文的目标是根据角色运动的交互作用来仿真流体，从而实现带有流体运动特征的高级运动特效。这类需要高细节层次角色动作影响的特效常常能被运用

基金项目：国家自然科学基金(60833007)，澳门大学基金。**徐添辰**，硕士，研究方向为实时计算机动画、虚拟现实，及非真实感渲染。E-mail：x-universe@live.com。**吴恩华**，博士，教授，研究方向为真实感图形生成、科学计算可视化，及虚拟现实等。E-mail：ehwu@umac.mo。

于武术、格斗、舞蹈，及其他带有艺术效果运动的虚拟展示。众所周知，角色动画本身就是一个复杂的处理过程，加之流体仿真与交互的要求，尽管之前也有许多成功的例子，但要准确实时控制角色每一寸皮肤与流体的交互，仍然是一个大挑战：需要跟踪运动轨迹，并寻找一个最合适的流体仿真技术。

流体仿真技术有两大类：基于网格（欧拉）空间与基于粒子（拉格朗日）空间。其中前者的方法，数据信息都严格按照空间顺序组织，无需搜索计算，然而在计算压强时，需要求解一个复杂的线性系统；后者中最典型的就是光滑粒子流体动力学（smooth particle hydrodynamics，SPH）[1]，由于数据都被载入到独立的粒子中，因此物理模拟和运动计算变得简捷，而由于粒子的空间索引关系未知，导致读取交互数据时需要搜索。

在此笔者选择SPH来实现本文的流体仿真，因为我们可以利用GPU进行通用计算，加速搜索，弥补其缺点。在CUDA和DirectX11的Shading Model 5中，GPU可以对缓冲进行随机读写，从而可以实现一些复杂的数组操作。基于GPU的这个能力，笔者提出了一个新的快速粒子搜索算法。

2 相关研究

本文主要涉及运动特效的生成渲染与流体仿真。其中运动特效的生成，主要针对类似用于运动模糊和速度线的几何轨迹跟踪方法；而对于流体仿真，笔者采用SPH的方法，其首先由Müller等[2]引进计算机图形学，用于基于粒子的交互式流体仿真。在此，笔者专注于SPH的效率瓶颈——粒子搜索算法。

2.1 运动特效

Sander等[3]利用GPU几何着色器和相邻图元拓扑结构，提出了一种快速遍历网格边缘的算法。其成果适用于实时运动模糊，针对在GPU单元中运动会模糊几何体的生成进行优化，有效地解决共享边缘重复挤压(extrusion)的问题。

此外，另一个由Xu等[4]提出的工作，其针对长时间运动轨迹的跟踪，提出了有效的宏观管线优化方案——分割轨迹法，并且将基于网格的流体仿真技术用于运动特效，求解简化的N-S方程，实现烟雾升腾的效果。这不乏为一次成功的尝试，但其结果比较单一，并缺乏立体的层次感。

2.2 用于SPH的粒子搜索

自SPH被运用于计算机图形学以来，不少研究工作是针对其效率的提升，如适应性采样[5]和预校正压强计算[6]等基于CPU的方法。随着GPU技术的广泛应用，Kipfer等[7]构建出一种粒子引擎，利用2D纹理来近似保存粒子的空间关系。Harada等[8]将桶排序和网格的思想引入了SPH搜索，但每个网格单元只能容纳固定数目的粒子。此外Zhou等[9]还提出了在GPU中构建KD-树结构，但这类层次性的数据结构及迭代性的链表结构暂时不适合在当前的GPU实现，高频率随机读写会将执行效率会大打折扣。

在Le Grand的宽阶段碰撞检测工作[10]中，他们首先将粒子位置与ID用散列编码，然后用双调排序等方法将散列排序，最后寻找每个网格单元的起始位置。这个算法的复杂度为 θ（NlogN），而且非常适合于CUDA和DirectX之类的基于GPU并行的工具实现。但是散列的编码，尤其是拥有大量粒子3D情况，需要超过32位的编码，使得散列的结构不再是一个简单的整型数据。本文的方法，在流程上与此类似，但无需散列构造和准确排序，并且提高搜索效率。

3 方法与原理

本文主要解决两个问题：如何根据角色的运动发射粒子；如何在角色运动的驱动下模拟流体。在本节，笔者将着重分析这两个问题的原理。

3.1 运动跟踪

为了解决特效与角色交互的问题，笔者首先分析角色的运动。的确，从每一帧角色动画数据中，我们可以获得其运动状态。然而，倘若从某个运动姿态的模型顶点直接发射粒子，这些粒子会呈很不均匀的片状分布。因此，需要先对角色运动进行轨迹跟踪。但又由于针对流体模拟的粒子寿命相对较长，像普通实现运动模糊等为目的几何轨迹跟踪难以达到维持较长的轨迹，而对整段运动进行跟踪又是一种浪费，于是笔者仅对当前帧和前一帧的运动状态进行跟踪和记录。

当获取当前关键帧和前一帧的角色状态后，需要一种几何模型来沿着轨迹生成粒子。其中较为直观的就是选择运动粒子模型（如图1(a)）：直接对处于当前帧和前一帧的运动状态中的顶点进行插值，获得新的顶点作为粒子。然而，实时动画中，原始模型的顶点往往很稀疏，直接按照运动粒子模型插值而得到的新粒子的分布必定也是相当稀疏，这样就很难符合流

体仿真的粒子条件。另一种选择是使用运动体模型（如图1(b)）：具有高密度的体积构造，但这种体积模型很难由简单地采样转化为粒子，其中牵涉到大量几何生成和采样。于是，笔者依旧基于运动粒子模型，并结合运动体模型对其进行改进，以解决粒子初始化时的分布问题。

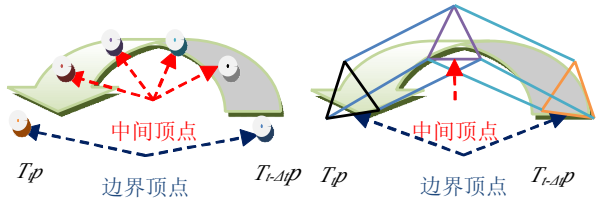

图1　(a)运动粒子模型；　(b)运动体模型

3.2　粒子发射端的分布问题

正如之前所述，由于运动粒子模型是根据原始角色模型的顶点分布经过插值生成粒子，并且原始模型的顶点分布往往很不均匀，这样会导致粒子的生成无法满足SPH的要求。更糟糕的是，过分依赖模型本身结构而生成粒子会导致粒子初始化不可控。

为了解决此问题，我们试图重新排布这些原始顶点，使其均匀分布。一个最便捷的办法就是利用角色模型的纹理坐标（简称UV坐标，往往纹理坐标都是均匀分布的）以及图形管线的光栅化处理。首先，需要填充一张位置缓冲，即将模型在当前运动状态下的表面位置信息展开，置入UV空间，生成一张纹理（位置缓冲）。通过光栅化，每个模型表面的像素都有一个对应的空间位置，而不仅仅是模型的原始顶点。如此，当发射粒子时，只要在位置缓冲的有效位置中随机选择一个位置，再与前一帧的位置插值，获得发射位置。

角色模型的纹理展开图中，并非全部是有效的像素。为了将粒子发射时选择有效的命中率提高至100%，可以首先将有效区域索引起来。笔者采用引入一张额外的二值蒙版（如图2所示，白色区域为有效区域），只要在预处理中遍历素有白色区域，将之纹理坐标索引并保存成一个列表即可保证粒子始终在有效区域内的采样。

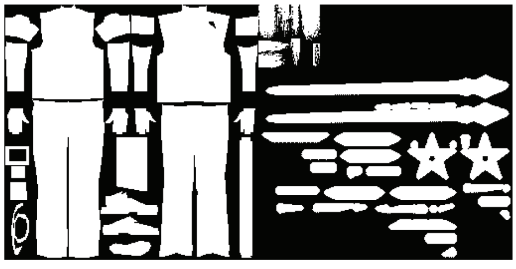

图2　针对位置缓冲的二值UV蒙版

3.3　光滑粒子流体动力学(SPH)

流体仿真通常遵守如图3所示的模拟循环。目标是模拟出流体中粒子的运动状态或者说是分布状态，而这个分布状态又取决于粒子当前的速度，速度又因为粒子受力情况而改变。粒子的受力情况通常分内力作用和外力作用。其中内力场又牵涉到粒子当前的分布状态，并且要考虑与其他粒子交互，是个相当复杂的问题。其中根据流体Navier-Stokes(N-S)方程(1) [11]，内力项基本由压强和黏度所提供。

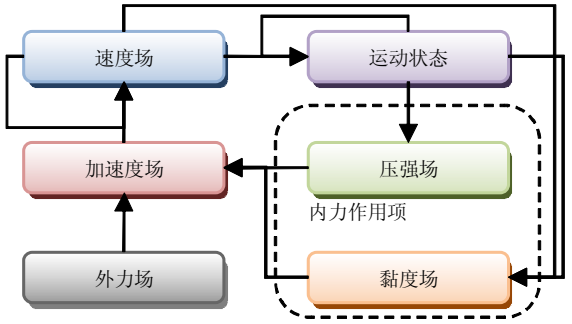

图3　流体模拟循环

$$\begin{cases} \rho\left(\dfrac{\partial \vec{u}}{\partial t}+\vec{u}\cdot\nabla\vec{u}\right)=-\nabla p+\mu\nabla^2\vec{u}+\vec{F} \\ \nabla\cdot\vec{u}=0 \end{cases} \tag{1}$$

u、p、ρ和F分别是速度，压力，密度和外力。所有的数据都用单位体积测量。其中$-\nabla p$ 和$\mu\nabla^2 u$分别为压强作用项和黏度项。直接对这两项求解需要解一个庞大的线性系统，可以采用雅可比迭代[12]或多网格处理[13]来求解，但这些依旧是复杂的工作，而本文采用SPH可以通过分布函数(2)计算密度场等来求解压强和黏度。

$$q_i=A_S(\vec{r}_i)=\sum_{\|\vec{r}_i-\vec{r}_j\|\le h} m_j\frac{q_j}{\rho_j}W(\vec{r}_i-\vec{r}_j,h) \tag{2}$$

其中q为密度、压强的梯度等数据信息，m和ρ分别为对应粒子的质量和密度，r为对应粒子的位置，h为光滑半径。

由此，密度场可以由下列公式计算：

$$\rho_i=\frac{315}{64\pi h^9}\sum_j m_j\left(h-\left\|\vec{r}_i-\vec{r}_j\right\|^2\right)^3 \tag{3}$$

然后根据密度场。可以计算出压强场：

$$p=\frac{mRT}{MV}=\left(\frac{RT}{M}\right)\left(\frac{m}{V}\right)=k(\rho-\rho_0) \tag{4}$$

其中$k=RT/M$是个常量，R、T、M分别是玻尔兹曼常量、温度和摩尔质量。

根据N-S方程，此时需要求解压强的梯度场。在

此，仅需要将梯度运算作用域光滑函数W即可获得压强所提供的加速度部分：

$$\vec{a}_{press}=\frac{45}{\pi h^6}\sum_j m_j\frac{p_i+p_j}{2\rho_j}(h-r)^2\frac{\vec{r}_i-\vec{r}_j}{r}$$
$$r=\left\|\vec{r}_i-\vec{r}_j\right\| \tag{5}$$

最后，黏度场的拉普拉斯也可以由(2)计算获得(6)，在此，内力作用项就都计算出来了。

$$\vec{a}_{visc}=\frac{45\mu}{\pi h^6}\sum_j m_j\frac{\vec{u}_j-\vec{u}_i}{\rho_i\rho_j}(h-r) \tag{6}$$

现在要考虑如何将轨迹段抽象成粒子、速度等数据放入N-S方程的外力项。外力可以根据运动粒子型轨迹段来计算获得。由于角色表面覆盖着刚发射的粒子，当粒子初始化后，作用域初始化粒子的外力，就会以内力形式传递给其他旧的粒子。所以，在此只需考虑如何根据运动轨迹初始化新粒子。由方程(7)可以计算获得初始化粒子的速度。

$$\vec{v}_{pre}=T(t+\tau)\vec{p}-T(t)\vec{p}$$
$$\vec{v}_{post}=T(t)\vec{p}-T(t-\tau)\vec{p}$$
$$\vec{v}_{impulse}=k\left(\frac{\vec{v}_{pre}+\vec{v}_{post}}{2\tau}\right) \tag{7}$$

其中 τ 是从t到t - ΔT时刻的插值参数。T(t)和p分别表示世界变换矩阵和轨迹上的样本位置。由此利用相邻轨迹顶点的位置信息变化，计算出了速度场的方向及大小。此外，k为能量传递率（由于考虑到能量损失）。

4 实 现

我们使用DX11来实现实验系统。对于几何原型的建立，该系统运用了分割轨迹法。然后将轨迹上的信息提取用来发射粒子，并用SPH模拟流体。

4.1 粒子沿轨迹的发射

首先我们将角色用高质量蒙皮技术进行权重混合变形，在此为了避免线性混合的瑕疵，采用双四元组混合技术[14]，但此方法加大了蒙皮的计算量，因此最好能够避免这种蒙皮运算的重复使用。幸运的是，在DX11的流水线渲染中，可以使用多步几何渲染技术：将GPU中蒙皮处理过的数据通过名叫流输出的功能返回顶点缓冲中储存，在下一个渲染步骤中，直接提取蒙皮后的数据进行渲染。

在蒙皮之后，处在当前运动状态的角色模型可以看作一个静态模型。然后，按照2.2节所述，将模型展开至UV空间储存位置信息，而获得一张位置缓冲纹理，如图4的例子所示。由于一个角色通常包含不止一张纹理，此时对于未知缓冲，需要合并下纹理，并加以简单的缩放（位置缓冲不需要原始漫反射纹理那么高精度）和偏移重新调整纹理坐标。在此，先定义两个二维向量：缩放因子和偏移因子。在图4的例子里，右半部分的左上角原本是256×256的头发纹理，此时将原始UV坐标以(0.125,0.25)缩放，再以(0.5,0)的偏移量平移，即可整合到512×256纹理。此时，位置缓冲的生成与采样均可以用以上方法从原始UV进行变换。

图4 根据图2的蒙版填充的位置缓冲纹理

当位置信息被保存后，连同前一帧的位置缓冲一起，我们可以随机选择一个有效位置，然后用随机参数将当前帧与前一帧的位置进行插值，得到粒子发射的位置。最后根据2.3节的(7)初始化粒子速度。整个粒子发射算法在DX11的计算着色器（compute shader）中执行，伪代码如下：

```
算法 1   CS_EmitMain(tid)
1:       if Particle[tid].life > lifelimit
2:           s ← ValidArea[random(0, Size(ValidArea))]
3:           p0 ← Position_t.load(s)
4:           p1 ← Position_t-Δt.load(s)
5:           τ ← random(0, 1)
6:           Particle[tid].pos ← τp0 + (1 − τ) p1
7:           prev ← (τ − Δτ)p0 + (1 − τ + Δτ) p1
8:           post ← (τ + Δτ)p0 + (1 − τ − Δτ) p1
9:           Particle[tid].velocity ← k(post - prev) / 2Δτ
10:          Particle[tid].life = random(0, lifelimit / 2)
11:      else Particle[tid].life++
12:      end if
```

4.2 流体运动模拟

为了实现快速高效的SPH流体模拟，笔者首先对流体模拟中的读写操作进行分析，并尝试简化模拟管线。表1陈列出了所有在模拟时所需的缓冲及其相关信息。其中Particle缓冲用于存储粒子的位置和速度，需要两个，在物理计算时一个作为整理和读取（ParticleS），一个作为写入（Particle）。然后，笔者在算法2中描述整个物理模拟的流程，其说明会在下一段具体解释。

（1）渲染至UV空间：记录角色当前的运动状

表1　SPH中用于物理计算的缓冲分配

缓冲	类型	数目
Position Buffer	纹理缓冲	2
Particle	结构缓冲	2
Density	结构缓冲	1
Acceleration	结构缓冲	1

算法 2　仿真时间间隔

1: 像素着色器: *Position*[*current*] ← RenderToUV(*mask*)
2: 计算着色器: *Particle*[*i*] ←Emit(*Position*)
3: 计算着色器: *ParticleS*[*i*] ← Arrange(*Particle*[*j*])
4: 计算着色器: *Density*[*i*] ←Density(*ParticleS*)
5: 计算着色器: *Acceleration*[*i*] ← Force(*Density*, *ParticleS*)
6: 计算着色器: *Particle*[*i*] ← Integrate(*Acceleration*[*i*], Const)

态，并存入当前帧的位置缓冲；

（2）推入粒子：根据算法1发射粒子，写入粒子缓冲；

（3）索引：根据粒子位置的顺序重新整理和索引粒子；

（4）密度计算：根据2.3节的方程(3)计算密度场；

（5）内力计算：根据2.3节中方程(4)计算压强，然后用(5)和(6)分别计算压强的梯度和黏度的拉普拉斯，以获得压强和黏度的作用项；

（6）集成：将内力项投影至加速度，并根据重力等外力约束更新最终加速度。最后更新粒子的速度和位置。

4.3　相邻粒子搜索

除了物理计算以外，相比基于网格的流体模拟手段，SPH的效率瓶颈在于光滑半径h内粒子搜索。一种选择就是直截了当地进行桶排序，然而每个网格单元（桶）中的粒子数分配是难以预测的，这样会导致空间浪费。而另一种策略是将粒子编码如散列进行全排序，但排序的代价依然比较高。值得注意的是，我们只关心不同网格单元粒子的顺序，而同一个网格单元粒子的顺序如何，是无所谓的。所以可以对粒子根据网格位置进行粗略的排序即可。

（1）首先分配若干缓冲来表示网格，如图5和表2所示。索引缓冲的x分量先用于统计每个网格单元的粒子数；偏移缓冲用于记录粒子相对于某个网格的顺序序号（单元偏移）；

（2）统计每个网格单元的粒子数目，并记录其单元偏移；

（3）计算网格单元中粒子段的首地址和尾地址，分别存于索引缓冲的x和y分量；

（4）按照网格单元首地址和单元偏移重整粒子顺序。

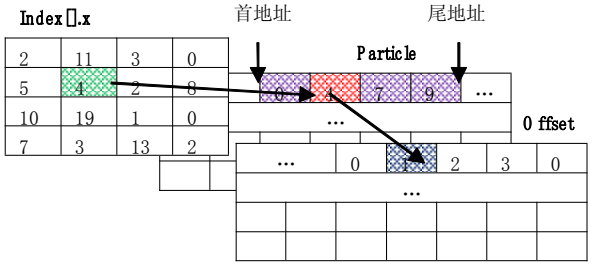

图5　缓冲使用

表2　用于粒子搜索的缓冲分配，其中也包含Particle

缓冲	类型	数目	规模
Particle	结构缓冲	2	粒子数
Offset	UINT	1	粒子数
Index	UINT2	1	网格单元数

值得注意的是，在GPU中进行统计每个网格单元中的粒子数目时，由于计数器 Index[].x是所有线程的共享资源，因此需要原子操作(atomics)来保证并发操作时的读写不可分。下列算法是实现这个环节的具体步骤：

算法 3　网格单元内粒子数统计

1: *pos* ← *Particle*[*i*].*position*
2: *j* ←PositionToIndex(*pos*)
3: Atom(*k* ← *Index*[*j*].*x*++)
4: *Offset*[*i*] ← *k*

统计完数目后，我们需要计算划分给每个网格单元粒子段的首尾地址。理论上，这本是一个递推的过程：

$$i^j_{end} = \begin{cases} a_j, & j = 0 \\ i^{j-1}_{end} + a_j & j > 0 \end{cases}$$
$$i^j_{start} = i^j_{end} - a_j \qquad (8)$$

其中，j为网格单元ID，a为网格单元内的粒子数。

如图6所示，我们的目标是将网格单元的粒子数累加起来，获得网格单元所对应的粒子段的尾地址。由于这个累加函数是递推的，在GPU上难以直接实现，需要将整个数据缓冲拆分，在送入GPU并行计算。下面就要以一个例子来描述如何利用GPU来并行完成这个累加。

正如图7所示，首先将整个缓冲分割，看作一个矩阵。针对每个矩阵的行，作为一个并行线程进行累加。然后提取结果中的做后一列，通过间隔索引再构成一个矩阵，做同样的行累加操作。依次类推，直到

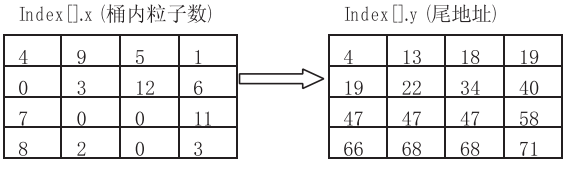

图 6　(a) 网格单元的粒子数；(b) 理想累加的结果

矩阵规模小于阈值为止。

当并行的矩阵行操作结束后，如图8所示，将除了最后一列与第一个元素以外的剩余元素和最后一列中上一行的元素进行相加，其中每个元素的加法都可以单独以一个线程并行执行。然后将结果作为上一级大矩阵的最后一列，再做相同的操作，直到原始缓冲内所有数据都被计算。

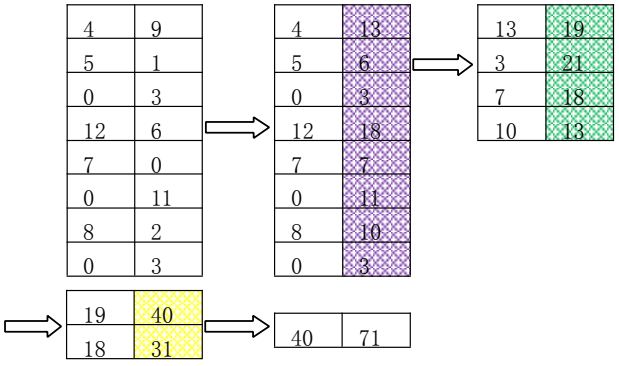

图7 并行线程内矩阵行累加

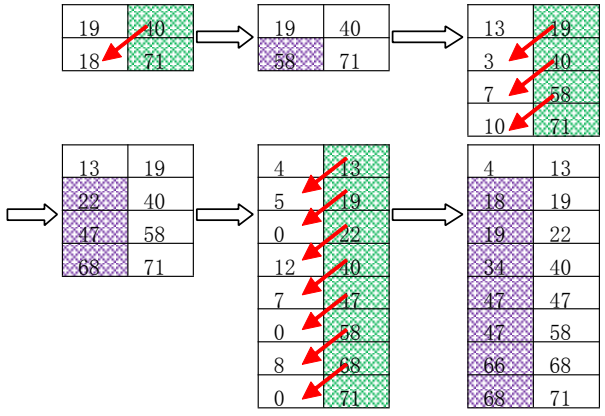

图8 对剩下元素的全局并行的地址偏移计算

这样，所有的粒子都可以根据对应的网格单元首尾地址和单元偏移进行重新排列，相同网格单元的粒子被排列到同一块粒子数据段内。因此，在类似求解密度场等计算中，只要根据近似光滑半径内的相邻网格单元ID进行索引即可获得单元内所有粒子数据。

5 结 果

在这一节，笔者评估本文的方法并且将它在各种复杂场景中对于各种运动效果的模拟结果逐个展示。测试机使用NVIDIA® Geforce GT240M和GTX590。

关于SPH流体仿真的搜索加速与之前的散列与双调排序作对比（如图9所示的帧数统计），本文方法的理论复杂度为 θ (N)，并且在空间上也是相对最优的，无需排序所用的临时缓冲。在GTX590等支持DX11全特性（主要包括原子操作）的机器上，可以完全用GPU实现流体的搜索索引、物理仿真和渲染。

此外，我们的模型用非常完整的方式结合了3D场景和基于SPH流体仿真的运动特效。这个例子（图10，模拟现今很多游戏中使用的各种技术，包括阴影映射、动态水面反射以及HDR等等。整个场景都是采用3D模型来搭建的一个相对完善的环境，表3对这个综合演示场景的数据量和帧速进行了统计。这种场景进行测试证明了我们提出的方法对于当前实时游戏引擎的适应性与可行性。其中水花和火焰的效果就是基于流体仿真的角色运动特效，飞溅和扩散的运动都具有高度细节的层次感。

6 结 论

本文呈现了一种基于SPH流体仿真，用于实时角色动画运动特效的手段。为了提供运动特效，我们拓展了GPU并行计算，成功地将粒子沿角色交互的轨迹发射，并通过SPH流体模拟来实现符合流体运动特征的效果。同时，我们也着重提升了流体仿真的效率，采用基于时间最优的桶排序搜索算法，并对空间也进

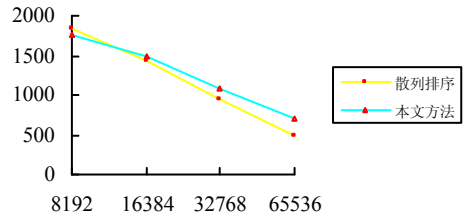

图9 与散列排序法对比帧速，粒子数：8K、16K、32K、64K

图10 综合场景演示，水火剑舞

表3 综合场景演示复杂度和帧速统计

项目	顶点数	面数
角色	25929	22054
粒子	65537	16388
场景	14530	14311
合计	105995	52749
机器	帧数（800×600，4xMSAA 反走样）	
GT240M	67.7 – 75.0	
GTX590	13.4 – 17.6	

行最优化，使结果符合普通用户级个人计算机的实时运行。

在未来的工作中，我们将进一步提升运动特效中流体的交互，尝试实现多种流体与多个角色混合的实时交互模拟。

参 考 文 献

[1] Monaghan J. Smoothed particle hydrodynamics [J]. Annual Review of Astronomy and Astrophysics, 1992, 30: 543-574.

[2] Müller M, Charypar D, Gross M. Particle-based fluid simulation for interactive applications [C] //Proceedings of SCA USA: ACM, 2003: 154-159.

[3] Sander P V, Nehab D, Chlamtac E, et al. Efficient traversal of mesh edges using adjacency primitives [C] //ACM Transactions on Graphics-Proceedings of ACM SIGGRAPH Asia 2008 (S0730-0301). 2008, 27(5): 144: 1-144: 9.

[4] Xu T C, Wu E H, Chen M, et al. Real-time motion effect enhancement based on fluid dynamics in figure animation [C] // Proceedings of VRCAI 2011. ACM, 2011: 307–314.

[5] Adam B, Pauly M, Keiser R, et al. Adaptively sampled particle fluids [J]. ACM Transactions on Graphics (S0730-0301) 26, 2007, 3: 48-54.

[6] Solenthaler B, Pajarola R. Predictivecorrective imcompressible SPH [J]. ACM Transactions on Graphics (S0730-0301) 28, 2009, 3, 40: 1-6.

[7] Kipfer P, Segal M, Westermann R. Uberflow: a GPU-based particles engine [C] //Proceedings of ACM SIGGRAPH/ Eurographics symposium on Graphics hardware. 2004: 115–122.

[8] Harada T, Koshizuka S, Kawaguchi Y. Smoothed particle hydrodynamics on GPUs [C] //Proceedings of Computer Graphics International, 2007: 63–70.

[9] Zhou K, Hou Q, Wang R, et al. Real-time KD-tree construction on graphics hardware [Z]. Technical report, Microsoft Research, 2008.

[10] Le Grand S. Broad-phase collision detection with CUDA [M]. GPU Gems 3, Chapter 32, NVIDIA, 2007.

[11] Pnueli D, Gutfinger C. Fluid mechanics [M]. NY(USA): Cambridge University Press, 1993.

[12] Golub G H, Van Loan C F. Matrix computations [M]. USA: Johns Hopkins University Press, 1996.

[13] Chentanez N, Müller M. Realtime eulerian water simulation using a restricted tall cell grid [C] //ACM Transactions on Graphics - Proceedings of SIGGRAPH 2011 (S0730-0301), 30, 4: 82.

[14] Kavan L, Collins S, Zara J, et al. Geometric skinning with approximate dual quaternion blending [J]. ACM Transactions on Graphics (S0730-0301). 2008, 27(4): 105:1-105:23.