清 华 大 学

综 合 论 文 训 练

题目：不可压平滑粒子流体动力学算法GPU并行加速及其应用研究

系　　别：软件学院

专　　业：计算机软件

姓　　名：费昀

指导教师：王斌 副教授

2013年 7月

中文摘要

近年来，随着硬件技术的发展、可编程着色器以及通用计算语言的出现使得在一定条件下图形处理单元的计算能力远远超过传统的中央处理器。因此，本文在使用图形处理单元对流体模拟与渲染进行加速的课题上进行初步的探索，其主要贡献包括：

1、提供了一套对预估校正不可压平滑粒子流体动力学算法的并行实现，用于不可压流体运动的交互模拟。

2、提供了一套平滑三维点云表面重建算法的并行实现，用于流体的实时渲染。

同时，本文的实现程序第一次将并行的三维不可压流体实时模拟、表面重建与渲染整合并开源，为后续研究与应用提供了一套有效的起步方案。为了体现其实际应用意义，本文展示该实现的两个相关应用：

1、通过引入基于几何的焦散与阴影，对三维水体进行高度真实的实时模拟与渲染。

2、通过引入随着温度而变化的粘度，以及流体实体结合，对中国糖画的交互生成进行模拟。

关键词：图形处理单元，硬件加速，并行计算，计算机动画，预估校正不可压平滑粒子流体动力学，实时物理模拟与渲染

ABSTRACT

Recently, the power of graphics processing unit (GPU) has been incredibly improved due to the development of hardware technology, in programming shading and in general purpose computing, which is much higher than traditional central processing units (CPU). Hence, in this thesis, we apply it to fluid simulation and rendering, which includes two main contributions:

1. a parallelized implementation of predictive-corrective incompressible smoothed particle hydrodynamics, which is a recently popular method for the simulation of incompressible fluid, producing real-time frame rate.

2. a parallelized implementation of smoothed surface reconstruction from 3D point cloud, for real-time rendering of the fluids motion.

Furthermore, the implementation accompanied with this thesis is the first time that the simulation of incompressible fluids, surface reconstruction and rendering are integrated into an open source package, which can be an effective start for further research and applications. To be concrete, we demonstrate two application of our fluid solver:

1. by introducing geometry-based caustics and shadows, we plausibly simulate and render the fluid in real-time.

2. by introducing temperate-related viscosity, and fluid-solid coupling, we simulate the painting process of Chinese sugar-coating in an interactive way.

Key words: graphics processing unit hardware acceleration parallel computing computer animation predictive-corrective incompressible smoothed particle hydrodynamics real-time physical-based simulation and rendering.

目 录

主要符号对照表

∇f	对函数f的梯度算子
$\frac{\partial f}{\partial x}$	对函数f在x分量上的偏导数
$\frac{\partial f}{\partial \vec{n}}$	对函数f在向量$\vec{n}$上的方向导数
$\nabla \cdot \vec{u}$	方向场$\vec{u}$的散度
$\nabla \times \vec{u}$	方向场$\vec{u}$的旋度
$\nabla \cdot \nabla f$	函数f的拉普拉斯算子
GPU	图形处理器 (Graphic Processing Unit)
CPU	中央处理器 (Central Processing Unit)
CUDA	统一计算设备架构 (Compute Unified Device Architecture)
SPH	光滑粒子流体动力学 (Smoothed Particle Hydrodynamics)
WCSPH	弱可压光滑粒子流体动力学 (Weakly Compressible Smoothed Particle Hydrodynamics)
PCISPH	预估校正不可压光滑粒子流体动力学 (Predictive-Corrective Incompressible Smoothed Particle Hydrodynamics)
Direct3D	微软三维图形加速平台
Shader	着色器，特指运行于GPU 上的代码
NS	纳维-斯托克斯 (Navier-Stokes)
FFT	快速傅里叶变换 (Fast Fourier Transform)
PN三角形	弯曲点法向三角形 (Curved Point-normal Triangles)

图 1　本文对水的实时模拟效果。

第 1 章　引言

1.1　研究背景

在生活中，流体是普遍存在的，每天人们都能接触到流动的水，腾升的烟雾，燃烧的火焰等等。近年来，随着软硬件的发展，实时真实感图形学已经成为高度产业化的领域。然而，对于流体的运动模拟，实时的技术产生的结果并不尽如人意，且与离线计算得到的结果有较大差距。

另一方面，在诸如水利、航天等领域，计算流体动力学已经成为一个单独而成熟的学科，然而它的主要目标与应用在于精确模拟流体的运动，而非使得用户得到快速近似且外观真实的结果。

本文所做的工作主要针对实时应用，包括计算机游戏、电影制作、以及医学训练等等，在尽可能精确的前提下，通过使用图形处理单元（GPU）对近年来最新的计算流体动力学方法进行加速，该方法称为预估校正不可压平滑粒子流体动力学（PCISPH）[1]。同时，为了真实地展现出流体的运动，本文使用GPU 对表面重建[2,3] 进行加速。

另外，本文展现实时流体模拟的两个应用：一个是较为传统的水的实时渲染，另一个是本文提出的新的应用即中国糖画的交互生成。在水的渲染中，为了使得流体具有尽可能真实的材质表现，本文提出一套基于几何体的实时焦散生成方法，该方法中可以精确模拟光线透过水面折射、经过底板漫反射再经过水面折射进入摄像机（LSDSE）的复杂路径。在糖画生成中，为了使得流体与固体进行有效交互，本文对固体表面进行蓝噪声采样并使用惩罚力的方式来防止流体穿透固体；同时为了模拟糖的温度黏度特性，本文对液体粒子计算温度随时间的扩散，并根据温度来改变液体的黏度。

1.2　研究现状

流体模拟是一个历史非常悠久的研究领域。一般认为，现代的流体动力学在19 世纪由克劳德-路易·纳维（Claude-Louis Navier）与乔治·加布里埃尔·斯托克斯（George Gabriel Stokes）创立，他们提出了著名的纳维- 斯托克斯方程组（Navier-Stokes equations，或简称为NS方程）。NS 方程描述了动量的守恒，

将它与另外的用于描述质量和能量守恒的方程联合起来后，流体的运动可以被完整的模拟。同时，为了计算的简便，通常仅使用这套方程描述牛顿流体，其中就包括了常见的液体与气体，如水和烟气等。

求解NS方程是一个非常复杂的问题。作为数学中最为难解的非线性方程之一，它自从提出以来，也大约只有70多个精确解被人们得到；同时，NS方程解的存在性与平滑性位列美国克雷数学研究所在2000年提出的7 个千禧难题之六。

因此，在工程应用上，人们通常使用数值计算方法来求得NS方程的数值解。这些方法通常分为两类：欧氏法（Eularian）以及拉氏法（Lagrangian）。这两类方法的主要区别在于使用了不同的思路对空间进行离散化。欧氏法将空间离散为均匀格子（Grid），并用有限差分或者有限元方法来计算各个变量随时间的变化。使用欧氏法可以非常容易地保证流体的不可压性，处理边界问题以及流体固体的结合，但由于在内存有限的情况下，格子的数目受到限制，从而限制了模拟空间的大小；并且由于差分精度不可能无限高，欧氏法存在难以解决的数值消散（numerical diffusion）问题，使得随着时间的发展，模拟的偏差逐渐增大。

拉氏法使用一套不同的思路来对空间进行离散化。在这类方法中，流体由点云组成，每个粒子都拥有自己的位置，并附带一些时间相关的属性，如速度、压强、密度、温度等等。在每一步迭代计算中，这些属性首先被计算出来，然后粒子的位置将得到更新，从而开始下一次迭代。拉格朗日法的主要优势在于：

1、由于这类方法自动保证质量守恒，同时不需要考虑对流问题，因此计算更简单，更易于并行计算。

2、不存在数值消散问题。

3、表面重建更容易。

4、由于不存在格子边界问题，流体可以自由在空间中运动。

因此，近年来，拉氏法逐渐流行起来，其中最受到关注的一种方法称为平滑粒子流体动力学（SPH）[4]。但这类方法也有相应的问题，其中最大的问题是如何维持流体的不可压性。2009 年，Solenthaler等人提出一种预估校正的方法（PCISPH）[1]对粒子的位置进行松弛，该方法有效地解决了均匀密度单相流中的不可压问题，同时继承了拉格朗日方法的种种优势。本文工作使用他们的

方法进行模拟，并将其并行化。

1.3 相关工作

1.3.1 平滑粒子流体动力学（SPH）

关于SPH的最初的研究工作始于Gingold、Monaghan 以及Lucy等人在1977年在天文学方面的工作[4,5]，用来模拟大范围的星际气体运动。在计算机图形学界，该方法一直没有受到太多的重视，直到2003年Müller等人的工作[6] 显示出SPH 方法非常适合实时或者交互式应用。然而，由于Müller 等人的工作并未能用SPH解决流体的不可压性，SPH方法模拟的质量一直以来被认为是比较低的。为了解决这个问题，Becker等人在2007年将Tait方程[7] 引入SPH 方法，实现了弱可压SPH[8]，或简称WCSPH。该方法部分解决了流体的不可压性，然而引入Tait 方程的代价也是很大的，为了维持流体的稳定性，模拟的时间步长也必须相应缩短到10^{-5}s 级，使得模拟的进程非常缓慢。为了进一步解决这个问题，Solenthaler 等人在2009年提出了预估校正不可压SPH[1]，即PCISPH，有效地解决了均匀密度单相流的不可压问题，模拟的步长也提升了两个数量级[9]。本文中涉及的所有液体均假设为均匀密度单相流，继承Solenthaler等人的方法，并通过GPU加速的手段完成一套高效率的实现。

1.3.2 GPU加速的实时流体模拟

由于流体模拟的历史非常悠久，完整的回顾超出了本文所关注的范围，因此这一小节中本文仅介绍较适于GPU并行化的方法。较为经典的方法通常仅具有二维流体（高度场）模拟的能力，这如Kass等人[10] 与Tessendorf等人[11]的早期工作即建立在二维高度场中使用快速傅里叶变换（FFT）求解波动方程的基础上。这些方法在之后随即被扩展到GPU上[12–14]。他们由于速度快，对硬件需求低，被广泛用于游戏与电影等业界的图形引擎中。

然而对于三维流体模拟的GPU加速方法还主要处于实验室阶段。由于拉氏法更易于并行[6]，因此GPU加速的相关工作主要集中在拉氏法上，或者说，在SPH上。一般认为，最初的GPU加速尝试由Amada等人[15]完成。在他们的工作中，SPH最耗时的一步，即点云的最近邻搜索，仍然在CPU上完成。之后，Harada等人[16]以及Zhang等人[17]实现了全GPU的SPH算法，一个覆盖全空间的网格结构被用于最近邻搜索：这种方法的劣势在于包含所有信息的全

空间网格非常消耗显存，并且很难做到高效实现。后续的一些工作对Harada等人的方法进行了改进，包括Le Grand[18]用于碰撞检测的工作后来也被用于SPH 中[19]，以及Goswami等人[20] 使用Z 索引[21]来进行最近邻搜索。最近的GPU最近邻搜索研究主要集中在近似方法上[22]，但由于物理正确性的要求，这些近似方法并不适用于SPH。在这些方法中，Le Grand的方法高效且易于使用Direct3D实现[19]，非常适合工程开发。因此本文使用他的方法进行最近邻搜索。

1.3.3 表面重建

本文中主要介绍与著名的立方体匹配（Marching Cubes）[23]相结合的技术，主要包括：早期由Blinn提出的滴斑法（Blobbies）[24]，主要缺陷在于无法产生平滑表面；Müller等人提出使用加权密度函数[6]，但仍然难以解决表面凹凸不平的问题；Zhu与Bridson[2]提出使用“平滑半径”的概念来解决该问题，但他们的方法会在凹处与水花之间产生瑕疵；Yu和Turk[25]提时出一种使用各向异性平滑核的方法，通过对相邻粒子的位置进行加权主成分分析（WPCA）得到粒子分布的主要方向轴，以此为依据对生成标量场时粒子间距离进行各向异性地放缩；最近，Solenthaler等人[26]以及Akinci等人[3]进一步改进了之前的方法，通过考虑相邻粒子间的位置梯度来消除瑕疵。他们的方法足以产生高质量的平滑表面。本文对以上方法[3,25]做一定的结合，使之适用于实时重建。

以上的方法通常被称为隐式表面重建（Implicit Surface Reconstruction），对应的另一类方法是显式（Explicit）表面重建，包括文献[27–30] 等之中的方法，这些方法需要精细的网格操作，并且难于并行化，更适用于离线应用。

此外，还有一类方法直接在屏幕空间进行操作，包括屏幕空间网格法（Screen-space Meshes）[31,32]、直接体渲染法（Direct Volume Rendering）[20]、以及表面溅射法（Surface Splatting）[33–35]等，尽管这些方法效率很高且易于并行，但其重建质量难以赶超隐式或显式表面重建法，另外，使用这些方法生成流体的阴影或者高质量的焦散也比较困难。

第 2 章　PCISPH方法理论、实现及其并行化

本章讨论实时PCISPH方法的基本原理与实现流程，其目的不仅仅是尽可能清晰地阐述其理论基础，同时也为读者提供详尽的实现依据，尤其关注与如何在GPU上实现一套高效、并行的PCISPH算法。

为了使读者更好地理解其物理意义以及理论基础，在第2.1 节，本文从不可压纳维-斯托克斯（NS）方程组的推导开始，展现其是如何从牛顿第二定律中推导出来的。之后，在第2.2 节中，本文暂且在忽略不可压性的情况下，介绍传统的SPH方法是如何来对NS方程进行求解。在第2.3 节中，本文讨论弱可压SPH （WCSPH）与PCISPH，完整地展现如何用PCISPH方法求解不可压NS 方程组。之后，本文在第2.4节中介绍对PCISPH并行化的方法，包括如何在GPU上通过基数或双调排序进行点云的最近邻检索。最后，为了体现本文的实际应用意义，本文在第2.5 节中介绍如何在GPU上进行光滑表面重建。

2.1　不可压纳维- 斯托克斯方程组

不可压纳维-斯托克斯方程由纳维与欧拉在1827年前后得出，最初称为欧拉方程，后经柯西于1828 年、泊松与1829 年、圣维南于1843 年，以及斯托克斯与1845 年分别以自己的方式对其进行修正，最终得到现在的形式：

$$\begin{cases} \rho(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u}) = -\nabla p + \eta \nabla \cdot \nabla \vec{u} + \rho \vec{g} \\ \nabla \cdot \vec{u} = 0 \end{cases} \tag{2-1}$$

其中第一道方程称为动量方程，第二道方程称为不可压条件。在本节其后的章节中，本文具体阐述每个符号以及整个方程组所代表的物理含义。

2.1.1　符号定义

本文首先对之后章节需要用到的一些多元微积分的知识进行简单的回顾，包括梯度、散度、旋度、拉氏算子以及高斯定理。为了便于阅读，本文对这些概念以及涉及的公式统一使用流体模拟相关文献常用的记号方式[36]。

梯度 计算一个函数的梯度就是计算它所有分量上的空间偏导数，组合为一个向量，即函数在该点的梯度。比如，在三维空间中，函数$f(x,y,z)$的梯度即写成：

$$\nabla f(x,y,z) = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}) \tag{2-2}$$

有时，梯度也单独作为算子存在，比如三维空间的梯度算子即写成：

$$\nabla = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}) \tag{2-3}$$

梯度代表了函数在空间某点各维度的变化率。而有时需要知道函数在特定方向上的变化率，那么就通过梯度计算出函数在这一点的方向导数，如函数f在方向$\vec{n}$上的方向导数记为：

$$\frac{\partial f}{\partial \vec{n}} = \nabla f \cdot \vec{n} \tag{2-4}$$

散度 散度算子用于描述向量场中的向量在某点的聚集或发散程度，在三维空间中，向量场$\vec{u}$的散度可以写成：

$$\nabla \cdot \vec{u} = \nabla \cdot (u,v,w) = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \tag{2-5}$$

注意虽然散度仅仅对向量场才有意义，但它本身是标量而不是向量。另外，散度符号之所以记成$\nabla\cdot$的形式是因为它也可以理解为梯度算子与向量场的点乘：

$$\nabla \cdot \vec{u} = (\frac{\partial}{\partial x}, \frac{\partial}{\partial y}, \frac{\partial}{\partial z}) \cdot (u,v,w) = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} \tag{2-6}$$

旋度 旋度算子用于描述向量场绕某点旋转的剧烈程度。在三维空间中旋度是一个向量，它的运算方式为：

$$\nabla \times \vec{u} = \nabla \times (u,v,w) = (\frac{\partial w}{\partial y} - \frac{\partial v}{\partial z}, \frac{\partial u}{\partial z} - \frac{\partial w}{\partial x}, \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}) \tag{2-7}$$

旋度为0的向量场也称为无旋场。

拉普拉斯算子 拉普拉斯算子又称拉氏算子，通常以梯度的散度形式存在，而在某些文献中亦写成∇^2或Δ 的形式，本文统一使用$\nabla \cdot \nabla$ 的形式。拉氏算子通常用于描述一个函数中某点的值与它邻域的平均值之差，或者说用于描述某点在它的邻域中是多么与众不同。在三维空间中，它的计算方式为：

$$\nabla \cdot \nabla f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} + \frac{\partial^2 f}{\partial z^2} \tag{2-8}$$

另外，拥有$\nabla \cdot (a\nabla f) = q$形式的方程称为泊松方程，特别的，当$q = 0$时，方程被称为拉普拉斯方程。

高斯定理 高斯定理是多元微积分中的基本定理之一，给定一个三维空间中有限区域Ω中的向量场$\vec{u}$以及包围区域的曲面的单位法向量$\vec{n}$，高斯定理通常记为：

$$\iiint_{\Omega} \nabla \cdot \vec{u} = \iint_{\partial\Omega} \vec{u} \cdot \vec{n} \tag{2-9}$$

也就是说，向量场$\vec{u}$在区域Ω中的空间积分等于$\vec{u}$在法向量$\vec{n}$上的投影在包围区域的曲面上的边界积分。对于流体而言，若$\vec{u}$为速度，那么高斯定理的意义便是：出入区域Ω的流量总和可以用流速在包围区域的曲面上的边界积分来描述。

2.1.2 纳维-斯托克斯方程推导

为了方便阅读，本文延续使用流体模拟文献常用的记号方式：使用$\vec{u}$来表示流体的速度，其三个分量为(u, v, w)；V用来表示体积；m表示质量；希腊字母$\rho = \lim_{\Delta V \to L^3} \frac{\Delta m}{\Delta V}$表示大于分子间距的小区间$L$中的密度，如水的密度$\rho = 1000kg/m^3$；字母$p = \lim_{\Delta A \to 0} \frac{\Delta F_p}{\Delta A}$表示压力$F_p$作用在小表面$\Delta A$上的压强；希腊字母$\eta$用来表示液体的粘度，或称为动力粘度，通常糖浆一类的液体具有高粘度，而酒精一类的液体具有较低的粘度。

2.1.2.1 动量方程

本节推导动量方程，由牛顿第二定律开始：

$$\vec{F} = m\vec{a} \tag{2-10}$$

牛顿第二定律表明物体的加速度$\vec{a}$由外力$\vec{F}$以及质量m决定。对于像SPH这样的粒子系统来说，每个粒子代表整个流体中的一小块，它拥有质量m、体积V以及速度$\vec{u}$。由于速度$\vec{u}$是空间及时间的函数，所以它的变化率即为：

$$\frac{d}{dt}\vec{u}(t, u, v, w) = \frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = \frac{\partial \vec{u}}{\partial t} + u\frac{\partial \vec{u}}{\partial x} + v\frac{\partial \vec{u}}{\partial y} + w\frac{\partial \vec{u}}{\partial z} \tag{2-11}$$

习惯上人们通常把上式即$\frac{d}{dt}\vec{u}(t, u, v, w)$记为$\frac{D\vec{u}}{Dt}$，称为向量场的物质导数（material derivatives）。

上式的第一项$\frac{\partial \vec{u}}{\partial t}$代表了当前流体在当前点的邻域的变化，而第二项$\vec{u} \cdot \nabla \vec{u}$代表了由于平流过程（各种属性随着流体而传输）带来的变动。将式2-11 带入式2-10可以得到：

$$F = m\frac{D\vec{u}}{Dt} = m(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u}) \tag{2-12}$$

又因为对于极小区域中的流体（粒子），有：$\rho = \lim_{\Delta V \to L^3} \frac{\Delta m}{\Delta V}$，因此式2-13 中的$m$可替换为$\rho$，即：

$$F = \rho(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u}) \tag{2-13}$$

下面对流体的受力F进行分析。对于单个粒子来说，它的受力可以分为其他粒子给予的内力F_I 以及整个流体受到的合外力F_E：

$$\rho(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u}) = F_I + F_E \tag{2-14}$$

最常见的外力就是重力，对于单个粒子来说，重力可以写成：$F_G = \rho \vec{g}$，代入则有：

$$\rho(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u}) = F_I + \rho \vec{g} \tag{2-15}$$

下面分析流体的内力F_I。首先假定本文面对的流体有不可压性（具体如何保证不可压将在下一节阐述）。这种不可压的粘性流通常也称为牛顿流体（Newtonian fluid），它的粘性压力与速度的梯度成正比[37]。同时由于流体不可压（速度场的散度为零，即$\nabla \cdot \vec{u} = 0$），整个流场不存在源或汇。当流体满足以上的条件时，就可以将流体的内力F_I分解为压强力F_P 以及粘性力F_V：

$$\rho(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u}) = F_P + F_V + \rho \vec{g} \tag{2-16}$$

其中粒子受到的压力只取决于粒子邻近的压强差：高压区域会对低压区域产生作用力。因此本文用压强的负梯度来计算压力，其方向从高压区域指向低压区域：

$$\rho(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u}) = -\nabla p + F_V + \rho \vec{g} \tag{2-17}$$

同时，由于本文假设流体不可压，于是粘性力则非常简单[37]，仅用粒子的速度与邻域速度的均差（即速度的拉普拉斯算子）即可描述，于是代入后得：

$$\rho(\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u}) = -\nabla p + \eta \nabla \cdot \nabla \vec{u} + \rho \vec{g} \tag{2-18}$$

这样便得到了式2-1中的动量方程。

2.1.2.2 不可压条件

事实上，任何真实的流体都是可膨胀可压缩的。流体的可压性也是声音传播的基础：人们能的一切听到声音都是流体的谐振（空气、水等）的结果。举水为例，很多时候人们会觉得水非常难以被压缩，然而，正是它的压强和密度变化使得人们可以在水下听到声音。

然而，正如人们平时所观察到的，流体的体积在日常情形下并不会大幅度地改变。比如，即使要对水进行很小幅度的压缩也需要非常强力的水泵，因此除了水下爆炸等极端现象以外，普通的外力并不足以显著改变水的压力。另外，最重要的是，尽管完全忽略流体的可压性会造成极度失真的结果，但精确模拟可压流也会使计算的复杂度显著增加。因此本文需要考虑可压性的模拟，但假设模拟的液体可压性为零，这样可以在不引入过多计算的情况下达成非常真实的模拟结果。

从数学上来说，流体的不可压性等同于在空间中的任何一点，它的邻域的体积不随时间发生任何变化，即：

$$\frac{d}{dt}V(\Omega) = \iint_{\partial\Omega} \vec{u} \cdot \vec{n} = 0 \tag{2-19}$$

由高斯定理可知，上式等价于：

$$\iiint_{\Omega} \nabla \cdot \vec{u} = 0 \tag{2-20}$$

由于上式对任意小的Ω均成立，对于拉氏法来说，即对具有任意V的粒子均成立，因此有：

$$\nabla \cdot \vec{u} = 0 \tag{2-21}$$

上式对空间中各点均成立。由于上式的形式，符合不可压条件的流场亦称为无散场。

2.2 光滑粒子流体动力学(SPH)

SPH是由Gingold和Monaghan[4] 以及Lucy[5] 与1977 年发展的一种数值计算流体运动的技术。在SPH中，对于流体使用点云来表示。点云中的每个粒子都

具有一定的影响范围。因此对于空间中的任意给定一点$\vec{r}$的任意物理量A而言，使用SPH 方法可以通过如下的离散求和来计算：

$$A(\vec{r}) = \sum_j A(\vec{r}_j) W(\vec{r} - \vec{r}_j, h) \Delta \vec{r} \tag{2-22}$$

其中$\vec{r}_j$是点$\vec{r}$邻域中第j个粒子的位置，$W(\vec{r}, h)$是一个径向对称平滑函数（也称为平滑核或者模糊核），其核半径为h。SPH 也有其他的几种解释：从粒子的角度看，SPH 方法相当于将每个粒子以半径h溅射（splat）到三维空间中；如果把点云看成多个冲击函数的叠加，那么SPH 方法相当于将这个函数与平滑核进行卷积操作。实际应用中，$W(\vec{r}, h) = W(-\vec{r}, h)$ 且$\int W(\vec{r}, h) d\vec{r} = 1$，同时$h$也被用于平滑核的支撑半径，即当$\|\vec{r}\| > h$ 时，$W(\vec{r}, h) = 0$。

2.2.1 求解纳维-斯托克斯方程

使用SPH方法模拟流体时，人们通常会用到以下几个物理量：对第j 个粒子，设其位置为$\vec{r}_j$，速度$\vec{u}_j$，质量m_j，密度ρ_j，体积$V_j = \frac{m_j}{\rho_j}$，则对于空间中任意一点$\vec{r}$，使用式2-22 计算其物理量：

$$A(\vec{r}) = \sum_j A_j \frac{m_j}{\rho_j} W(\vec{r} - \vec{r}_j, h) \tag{2-23}$$

为了计算简便，本文直接设每个粒子所代表的质量相同，即$m_j = 1$（此时密度ρ也称为数量密度，以区域内粒子总数衡量），于是有：

$$A(\vec{r}) = \sum_j \frac{A_j}{\rho_j} W(\vec{r} - \vec{r}_j, h) \tag{2-24}$$

SPH的一大优势在于物理量的空间导数可以直接通过平滑核的导数的加权和得到[6]，即：

$$\nabla A(\vec{r}) = \sum_j \frac{A_j}{\rho_j} \nabla W(\vec{r} - \vec{r}_j, h) \tag{2-25}$$

以及：

$$\nabla \cdot \nabla A(\vec{r}) = \sum_j \frac{A_j}{\rho_j} \nabla \cdot \nabla W(\vec{r} - \vec{r}_j, h) \tag{2-26}$$

根据上面的规则，可以得到各个物理量的计算公式，如对于粒子i，它的密度可以通过邻域的加权和计算：

$$\rho(\vec{r}_i) = \sum_j W(\vec{r} - \vec{r}_j, h) \tag{2-27}$$

压力：

$$F_P(\vec{r}_i) = -\nabla p(\vec{r}_i) = -\sum_j \frac{p_j}{\rho_j} \nabla W(\vec{r}_i - \vec{r}_j, h) \tag{2-28}$$

由于上式对于两个粒子i和j不一定对称，不符合牛顿第三定律，因此按照Müller等人的建议[6]，上式改为：

$$F_P(\vec{r}_i) = -\nabla p(\vec{r}_i) = -\sum_j \frac{p_j + p_i}{2\rho_j} \nabla W(\vec{r}_i - \vec{r}_j, h) \tag{2-29}$$

粘性力：

$$F_V(\vec{r}_i) = \eta \nabla \cdot \nabla p(\vec{r}_i) = -\sum_j \frac{\vec{u}_j}{\rho_j} \nabla \cdot \nabla W(\vec{r}_i - \vec{r}_j, h) \tag{2-30}$$

同压力一样，为了满足牛顿第三定律，上式改为：

$$F_V(\vec{r}_i) = \eta \nabla \cdot \nabla p(\vec{r}_i) = -\sum_j \frac{\vec{u}_j - \vec{u}_i}{\rho_j} \nabla \cdot \nabla W(\vec{r}_i - \vec{r}_j, h) \tag{2-31}$$

2.2.2 平滑核

平滑核的选择对于模拟的速度、稳定性以及质量影响都非常之大。对于这一点，Müller 等人[6] 提供了一套非常有效的平滑核，分别用于不同的物理量的计算，他们的方法也被目前大部分关于SPH的工作沿用。

其中第一个平滑核用于插值密度ρ，称为Poly6核（$r = \|\vec{r}\|$，图 2.1 左）：

$$W_{poly6}(\vec{r}, h) = \begin{cases} \frac{315}{64\pi h^9}(h^2 - r^2)^3 & 0 \le r \le h \\ 0 & otherwise \end{cases} \tag{2-32}$$

它的梯度为：

$$\nabla W_{poly6}(\vec{r}, h) = \begin{cases} -\frac{945}{32\pi h^9}(h^2 - r^2)^2 \vec{r} & 0 \le r \le h \\ 0 & otherwise \end{cases} \tag{2-33}$$

拉氏量为：

$$\nabla \cdot \nabla W_{poly6}(\vec{r}, h) = \begin{cases} \frac{945}{8\pi h^9}(h^2 - r^2)(r^2 - \frac{3}{4}(h^2 - r^2)) & 0 \le r \le h \\ 0 & otherwise \end{cases} \tag{2-34}$$

由于Poly6核的梯度在中心变为0，因此它不适用于插值压力，为了使粒子接近时具有较大的压力，必须使用另一种梯度在0 点有较大取值的平滑核来插值压力，称为Spiky核（图 2.1 中）：

$$W_{spiky}(\vec{r}, h) = \begin{cases} \frac{15}{\pi h^6}(h - r)^3 & 0 \le r \le h \\ 0 & otherwise \end{cases} \tag{2-35}$$

它的梯度为：

$$\nabla W_{spiky}(\vec{r}, h) = \begin{cases} -\frac{45}{\pi h^6 r}(h - r)^2 \vec{r} & 0 \le r \le h \\ 0 & otherwise \end{cases} \tag{2-36}$$

由于Poly6核的拉氏量在接近0点时很快变为负值，使用它对粘性力插值将会对粒子起到加速的作用，这与粘性力降低粒子速度的特性不符，因此需要使用另一种平滑核插值粘性力，称为粘度核（图 2.1 右）：

$$W_{viscosity}(\vec{r}, h) = \begin{cases} \frac{15}{2\pi h^3}(-\frac{r^3}{2h^3} + \frac{r^2}{h^2} + \frac{h}{2r} - 1) & 0 \le r \le h \\ 0 & otherwise \end{cases} \tag{2-37}$$

它的拉氏量为：

$$\nabla \cdot \nabla W_{viscosity}(\vec{r}, h) = \begin{cases} \frac{45}{\pi h^5}(1 - \frac{r}{h}) & 0 \le r \le h \\ 0 & otherwise \end{cases} \tag{2-38}$$

2.3 预估校正不可压SPH

这一节阐述如何在保证流体不可压性的前提下计算压强p。为了计算简便以及易于理解，本文使用一种不同但等价于Solenthaler 的文章[1]的方式来进行阐述。

已知流体在不可压的情况下有：

$$\frac{d}{dt}V(\Omega) = 0 \tag{2-39}$$

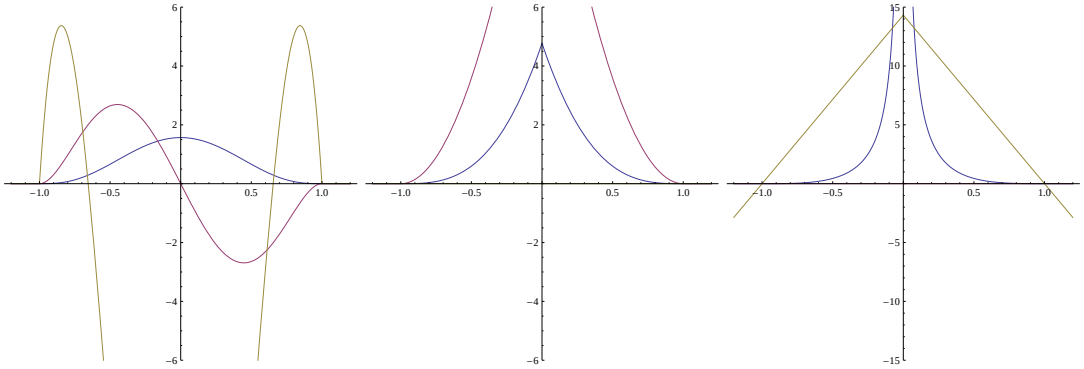

图 2.1　三种平滑核。蓝线：平滑核自身；绿线：梯度；红线：拉氏量

上式对任意点的任意邻域Ω均成立。由$V = m/\rho$以及m恒定可知，上式对于SPH等同于：

$$\frac{d}{dt}\rho(\Omega) = 0 \tag{2-40}$$

由式2-27可知，对于粒子i，有$\rho(\Omega) = \rho_i$，带入上式即得：

$$\frac{d}{dt}\rho_i = 0 \tag{2-41}$$

对于静止液体，存在一静止密度ρ_0，如对于水来说，$\rho_0 = 1000kg/m^3$。由式2-42可知，若要使流体不可压，就需要使得粒子的密度变化率为0，于是就可以设上一时间点的粒子密度总是等于ρ_0，这样便有：

$$\Delta\rho = \frac{d}{dt}\rho_i = \rho_i - \rho_0 = 0 \tag{2-42}$$

为了找到能满足或近似满足这个条件的压强p_i，本文用一种迭代松弛的方法：首先随意设定一个压强p，然后在迭代的每一步计算$\Delta\rho$，然后根据$\Delta\rho$对p_i进行调整，Solenthaler等人发现[1]，对压强的调整与$\Delta\rho$呈正比例关系，即每一步迭代中：

$$p_i += \delta\Delta\rho_i \tag{2-43}$$

在实际实现时，本文使用系数$\delta = 1$，即可取得很好的效果。

预估校正不可压SPH的算法流程如下：

1. 对于全部的粒子建立空间索引
2. 对于全部的粒子计算粘性力、合外力，并根据这些力的作用计算粒子的加速度。

3. 根据粒子的加速度与时间步长得到下一时刻的速度和位置。
4. 初始化全部$p_i = 0$，$F_P = 0$
5. 计算粒子在加有当前压力的下一时刻加速度、速度和位置。
6. 计算粒子在这样的位置下的邻域密度ρ_i，以及密度差$\Delta\rho = \rho_i - \rho_0$
7. 根据$\Delta\rho$对压强p_i进行修正，并计算新的压力F_P
8. 若对任意粒子i有$\rho_i - \rho_0 > \epsilon$则返回第5 步，否则结束迭代，使用当前压力计算下一步加速度、速度和位置并启动再下一时刻的计算。

在实际实现中，为了提高效率，本文还做了一些额外的优化，如利用时域连续性，使用上一时刻的压强作为当前时刻的初始化值，这样可以减少迭代次数。同时采取每N帧完整迭代K次，而N帧之间仅迭代K/M 次等策略，本文使用$N = 10, K = 8, M = 2$，可在质量和效率之间取得较好的平衡。

2.4 SPH并行化

这一节介绍如何将上面所述的算法进行并行化。本文的并行化采用一个线程对应一个粒子的思路。由于对任意一个SPH 的粒子计算各个物理量时通常都要搜索它周围离这个粒子最近（平滑核支撑半径范围）的所有其他粒子，并通过平滑核进行插值，因此SPH并行实现中最影响性能的便是最近邻搜索的快慢。简单的N^2 搜索（即对任意粒子，遍历空间中所有其它粒子）的做法的效率是非常低下的，因为粒子的邻域（平滑核支撑半径）对于整个空间来说是很小的，即对于单个粒子而言，只有很小的一部分其它粒子需要被遍历。本文工作沿用Direct3D SDK中的实现[19]。该实现使用Le Grand 的方法[18]，通过建立有序的空间索引来加速每次最近邻搜索的过程。

2.4.1 空间索引

对于存储在显存中数组$Array_{particle}$ 中的粒子，建立空间索引的步骤如下（见图 2.2）

1. 把整个空间划分为均匀的格子，对于每个格子指定一个索引号，这个索引号与格子的空间位置相关，一种简单的索引号可以是格子的坐标点乘空间划分的大小，如在256^3大小的空间划分中，坐标$(0x2, 0x3, 0x4)$的格子即有$(0x2, 0x3, 0x4) \cdot (0x100, 0x100, 0x100) = 0x020304$的索引号。当然也可以

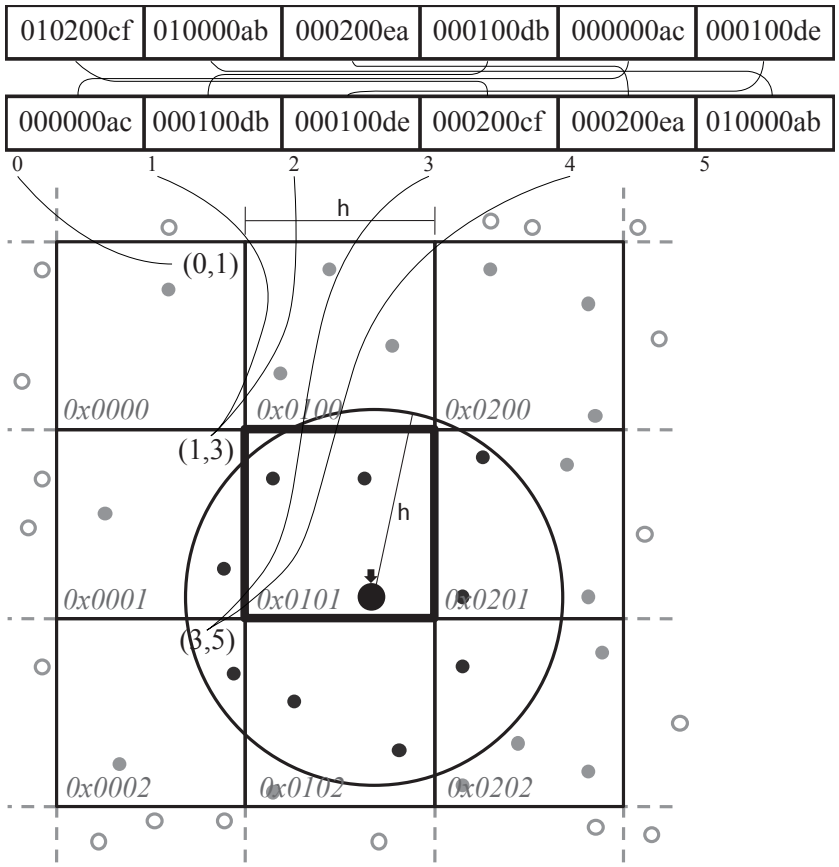

图 2.2　建立空间索引。黑点：对当前粒子有影响的粒子；灰点：对当前粒子无影响的粒子。（图片更改自参考文献[38]）

用其它的索引号计算方法，如Goswami 等[20] 提出用Morton编码[21] 可以更有效。

2. 对每个粒子根据位置计算其所属的格子以及对应的索引号，将索引号（一个32位整型变量）置于高位（在后）、粒子在$Array_{particle}$中的下标（另一个32 位整型变量）置于低位（在前）连成一个64位长整型数，存储得到索引号数组$Array_{idx}$。
3. 对$Array_{idx}$进行排序。由于格子索引号被置于高位，因此排序后处于一个格子内的所有粒子在$Array_{particle}$ 中的下标将连续地存储在显存中。
4. 对排序后的$Array_{idx}$中的每个值判断是否是一个格子的开始（或结束）。这可以通过当前值的高32 位与数组里前（或后）一个值的高32位是否一致来判断，如若不一致，则说明当前值是一个格子的开始（或结束），将当前值在$Array_{idx}$ 里的下标记录于一个空间划分大小的3D纹

理$Array_{sup}$中。

这样便建立好了空间索引，下面首先详述对$Array_idx$进行排序时所用算法，然后介绍如何使用空间索引进行最近邻检索。

2.4.1.1 GPU上的排序

并行排序算法的历史可以上溯到上世纪80 年代，那时很多算法就已经被实现在大型向量机上[39]（如著名的CRAY-1 等）。由于这些向量机在硬件上与GPU具有类似的逻辑结构，因而很多算法被沿用至今。这些算法中最常用的是并行双调排序与基数排序。

双调排序 首先介绍并行双调排序。与其它排序算法相比。它的GPU实现具

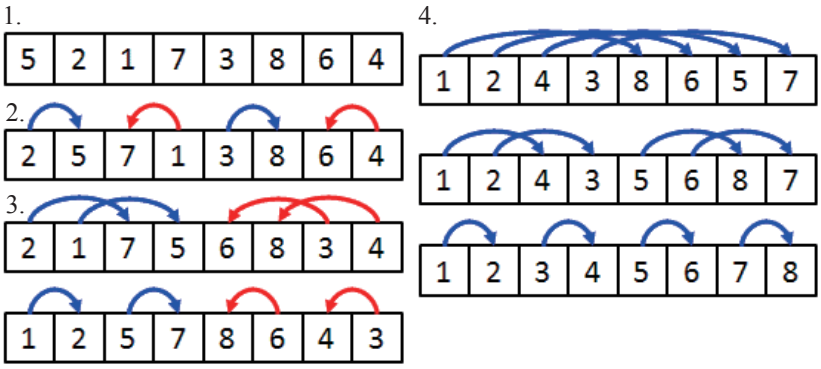

图 2.3　并行双调排序（图片来自Direct3D SDK文档[40]）

有简单便捷的特点。它的步骤如下（见图 2.3）：

1. 间隔每两个元素对相邻元素分别由小到大（蓝色）及由大到小（红色）排序。
2. 间隔每四个元素对隔二元素分别由小到大（蓝色）及由大到小（红色）排序，然后间隔每四个元素对相邻元素分别由小到大（蓝色）及由大到小（红色）排序。
3. 间隔每八个元素对隔四元素分别由小到大（蓝色）及由大到小（红色）排序，然后对隔二元素分别由小到大（蓝色）及由大到小（红色）排序，再对相邻元素分别由小到大（蓝色）及由大到小（红色）排序。
4. 以此类推，直到排序完成。

对N个元素使用N个线程，则它的完全并行时间复杂度为$O(log_2^3 n)$。

基数排序 基数排序是最古老的排序算法之一。它把待排序的n 位键值分成b 位的数串并从低到高排序n/b次。对于一个给定数位的数串，统计该数位上比它小的所有数串的个数；当所有数串处理完后，该数串之前数串的个数将是这个数串在输出的数组中的索引（也称作“ 阶”）；在得到所有数串的索引后，将每个数串输出到索引指定的位置上。

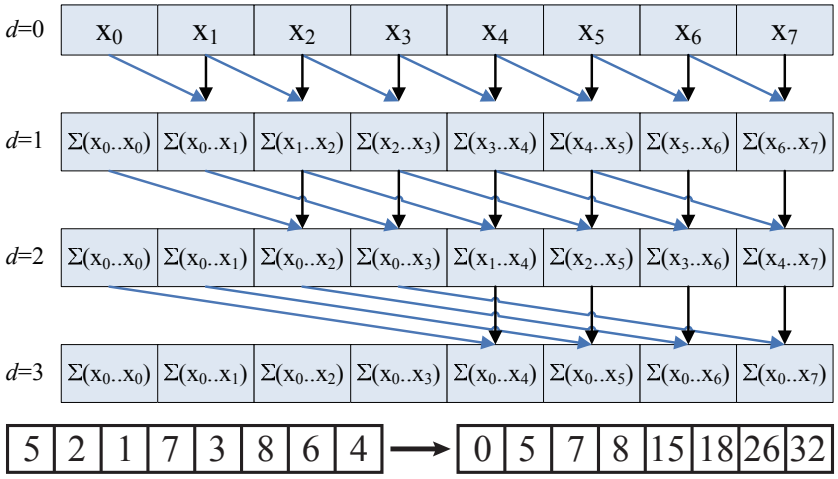

图 2.4　并行前缀和示意图，图片更改自Harris等人的文章[41]。

基数排序是一种相当容易并行的算法，因为统计数串个数的过程可以使用一次并行独占前缀和（exclusive prefix sum，也称为并行独占扫描）来完成（见图 2.4）。并行基数排序的一次迭代步骤如下[42]：

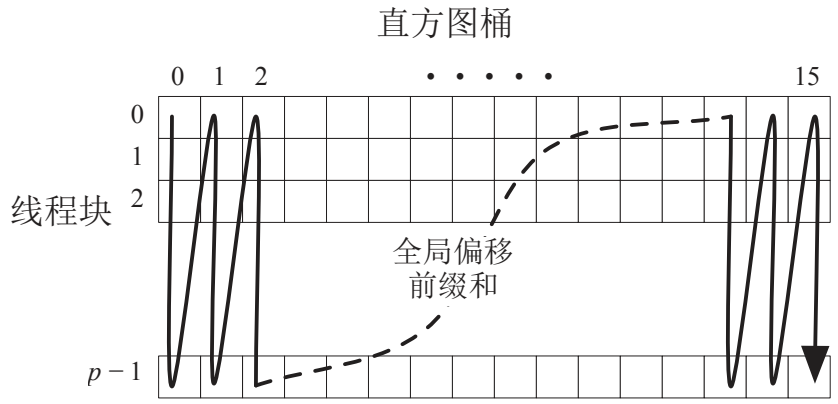

图 2.5　计算全局数位偏移示意图，图片摘自Satish 等人的文章[42]。

1. 将线程分组，每组线程将数据读入片上缓存并排序。
2. 统计数串的直方图，并与分组排序后的结果存入显存。

3. 对列主序存储的直方图计算前缀和，从而计算出全局的数位偏移（见图2.5）。
4. 利用上一步结果，每组线程将它排序后的结果复制到正确的位置。

在实际应用中，为了开发简便，本文实现调用高度优化的 Thrust 库[43]中的 stable_sort 函数来实现并行基数排序。

2.4.1.2 最近邻检索

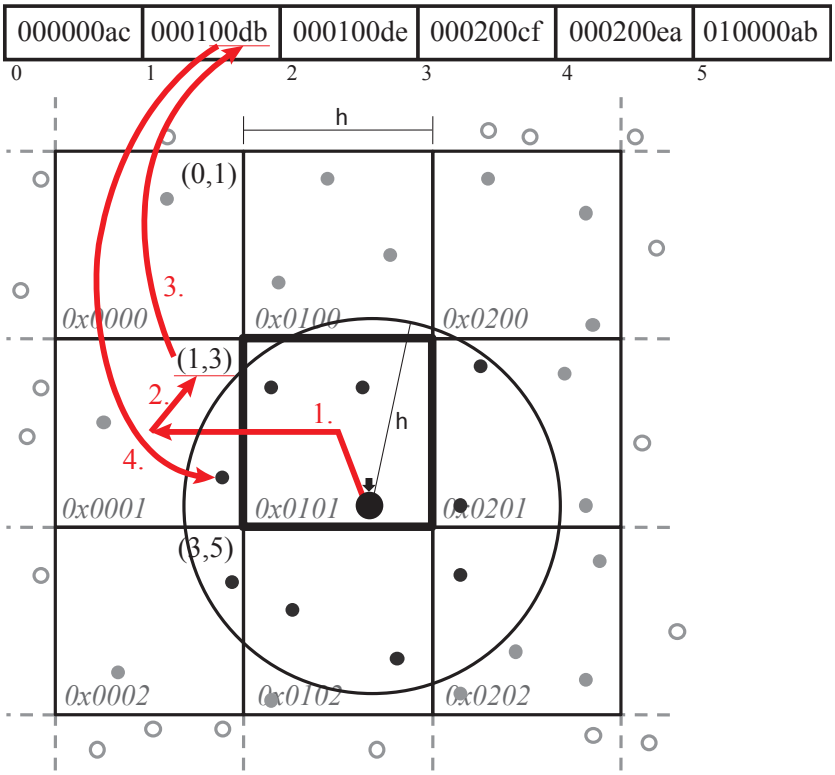

图 2.6 通过空间索引找到临近点。

在建立了空间索引之后便可以开始对各个物理量进行计算。在计算的时候会进行最近邻搜索，分为4步（见图 2.6）：

1. 计算当前粒子在空间划分中的位置，搜索其自身以及前后左右上下共27个邻域。
2. 对每一个相邻的划分，读取其对应的$Array_{sup}$中的起始下标和结束下标。
3. 从起始下标遍历到结束下标，根据下标读取$Array_{idx}$中的数据，截取低位，得到粒子在$Array_{particle}$中的下标。
4. 根据下标读取$Array_{particle}$ 中的位置和物理量信息，进行计算。

2.5 表面重建

为了得到真实的渲染结果，每进行一步的SPH模拟都要将点云转换为可着色的表面。本文的表面重建的步骤如下：

1. 对于每个粒子i计算其位置均值$\overline{x_i}$。为了增强时域连续性，将结果与上一时刻的位置均值做线性平均（此方法称为XSPH[44]）。
2. 根据$\overline{x_i}$及其邻域的粒子位置进行加权主成分分析，得到代表粒子邻域分布形状的矩阵$\mathbf{C_i}$，并根据$\mathbf{C_i}$及其特征值计算放缩标量S_i。
3. 对于三维密度场中每个体素，搜索其邻域的粒子j，根据体素中心到粒子距离进行插值得到密度（如图 2.7显示了将三维密度场以纵轴二维切片形式可视化的结果，亮度即为密度值）。插值时将距离r_{ij}除以S_j进行调整，然后与2倍平滑核半径$2h$进行比较，若有$r_{ij}/S_j < 2h$则进行插值。
4. 对三维密度场进行立方体匹配，得到三角形网格。
5. 对三角形网格进行表面细分，生成PN三角形，得到光滑的网格。
6. 在对网格着色前，在屏幕空间对网格的法线进行双边滤波，得到光滑的表面。

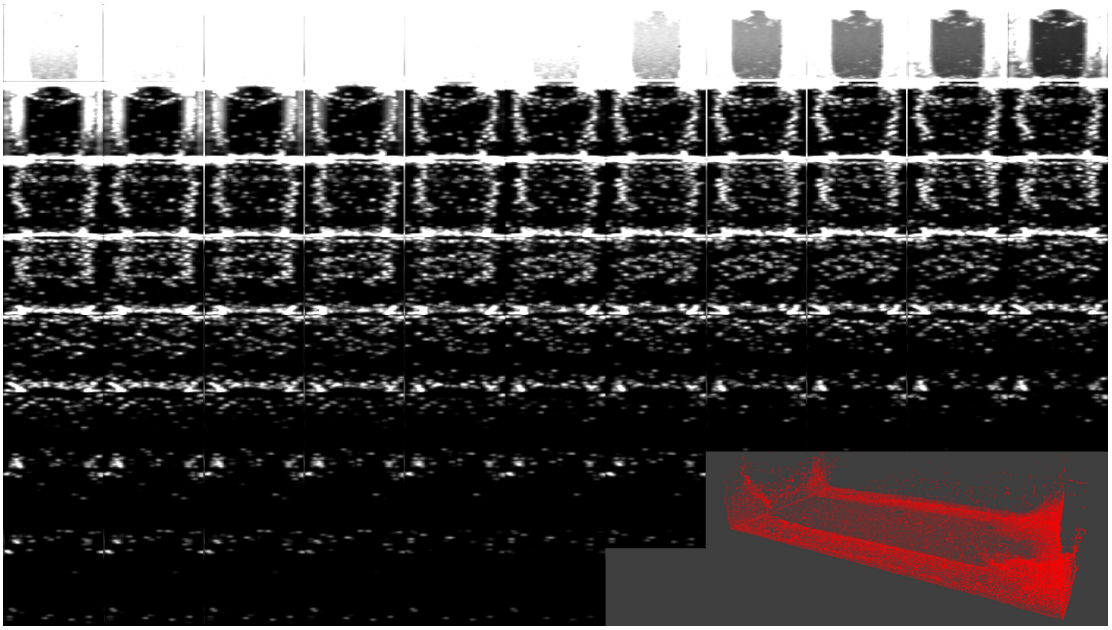

图 2.7　对生成的三维密度场进行纵轴二维切片可视化的结果，亮度即为密度值；切片以在三维空间从低到高的顺序从左到右、从上到下依次摆放。右下方显示了点云分布。

2.5.1 放缩标量

本文沿用Yu和Turk的工作[25]对邻域粒子位置分布进行分析。他们的方法使用加权主成分分析（WPCA）[45]来确定邻域粒子位置分布的各向异性程度。同时本文结合Zhu以及Akinci等人的工作[2,3] 计算放缩标量。其方法的步骤如下：

1. 计算邻域粒子位置的加权平均：

$$\vec{x}_i^w = \sum_j W_{PCA}(\vec{x}_j - \vec{x}_i, h)\vec{x}_j / \sum W_{PCA}(\vec{x}_j - \vec{x}_i, h) \tag{2-44}$$

其中:

$$W_{PCA}(\vec{r}, h) = \begin{cases} 1 - \frac{r^3}{h^3} & 0 \le r \le h \\ 0 & otherwise \end{cases} \tag{2-45}$$

2. 根据加权平均计算方差矩阵：

$$\mathbf{C_i} = \sum_j W_{PCA}(\vec{x}_j - \vec{x}_i, h)(\vec{x}_j - \vec{x}_i^w)(\vec{x}_j - \vec{x}_i^w)^T / \sum W_{PCA}(\vec{x}_j - \vec{x}_i, h) \tag{2-46}$$

3. 使用Kopp的方法[46]对$\mathbf{C_i}$进行快速奇异值分解，得到特征值三元组$\mathbf{\Sigma_i}$：

$$\mathbf{C_i} = \mathbf{R_i \Sigma_i R_i}^T, \mathbf{\Sigma_i} = diag(\sigma_0, \sigma_1, \sigma_2) \tag{2-47}$$

4. 计算最大特征值与中间特征值的比值$\gamma_i = max(\mathbf{\Sigma_i})/median(\mathbf{\Sigma_i})$，限制其值于[1,$\gamma_{high}$] 之间，然后由$\gamma_i$ 计算放缩标量S_i:

$$S_i = \alpha(1 - (1 - (\frac{\gamma_{high} - \gamma_i}{\gamma_{high} - 1})^2)^3) + \beta \tag{2-48}$$

在本文的实现中，常量γ_{high}=4，$\alpha = 0.75$，$\beta = 0.25$。

在得到每个粒子所需的S_i后便可以计算三维密度场，对于每个体素，本文以其中心为搜索的中心，$2h$为搜索半径，得到所有附近粒子的位置并使用三次B样条曲线[47]进行插值：

$$\phi(\vec{r}_i) = \sum_j W_{B-cubic}(\frac{\vec{r} - \vec{r}_j}{S_j}, h) \tag{2-49}$$

其中

$$W_{B-cubic}(\vec{r}, h) = \begin{cases} \frac{1}{\pi}(1 - \frac{3r^2}{2h^2} + \frac{3r^3}{4h^3}) & 0 \le r < h \\ \frac{1}{4\pi}(2 - \frac{r}{h})^3 & h \le r \le 2h \\ 0 & otherwise \end{cases} \tag{2-50}$$

2.5.2 移动立方体

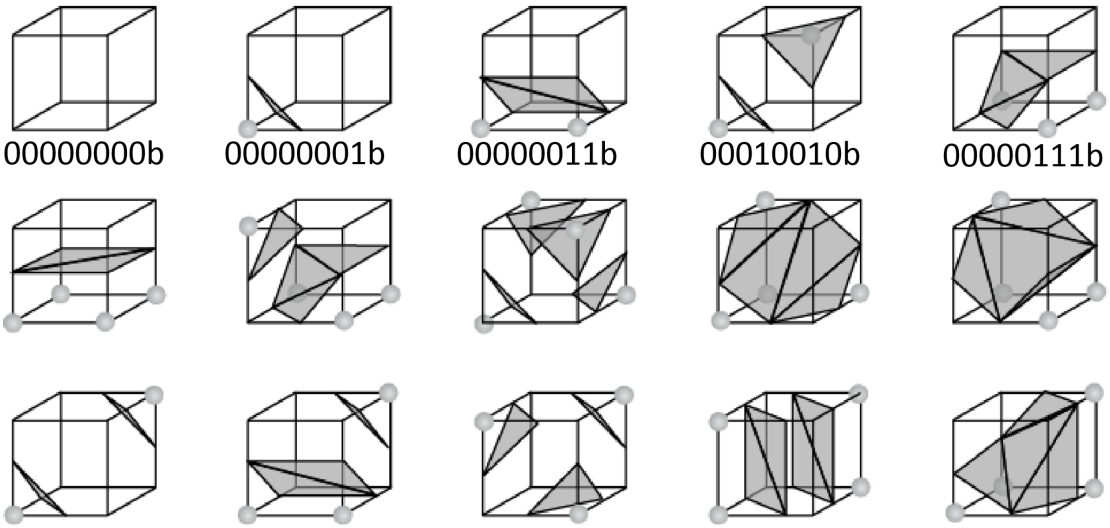

图 2.8 移动立方体示意图。图片更改自参考文献[23]。

移动立方体[23]是一种经典的从密度场生成三角形等值面的技术。它把密度场均匀划分为格子，对格子的胞元的每个角点的密度场进行评估，根据其是否大于某个常量阈值来决定生成三角形的形式。它的步骤如下：

1. 检查胞元立方体的12条边是否与等值面相交：如果边的两个端点一个的密度小于阈值且另一个的密度大于阈值则判为相交，否则则判为不相交。每条相交的边都生成一个对应的三角形顶点。
2. 通过胞元的8个角点是否大于等值面生成一个8位的索引（大于等值面该位则为1，否则为0）。每个索引对应一种三角化的模式，因此通过索引便可得到哪些三角形顶点相连（如图 2.8）。
3. 对密度场进行差分[20] 从而生成法线：

$$\vec{n} = \nabla\rho_{x,y,z} = \begin{pmatrix} \frac{\partial\rho}{\partial x} \\ \frac{\partial\rho}{\partial y} \\ \frac{\partial\rho}{\partial z} \end{pmatrix} \approx \begin{pmatrix} \rho_{x+1,y,z} - \rho_{x-1,y,z} \\ \rho_{x,y+1,z} - \rho_{x,y-1,z} \\ \rho_{x,y,z+1} - \rho_{x,y,z-1} \end{pmatrix} \tag{2-51}$$

为了得到光滑的法线，在对密度场采样的时候使用三线性插值。

2.5.3 表面平滑技术

在立方体匹配中，对于离散格子的每个胞元都要进行计算。这样由于硬件

性能所限，对空间的划分精细度不可能无限高，因此生成的网格仍然不够光滑（见图 2.11）。在高分辨率下放大渲染的时候，这种局限性会导致明显的多边形瑕疵。

为了进一步改进渲染的质量，本文使用表面镶嵌与细分技术得到更为细密的三角形网格，并通过在屏幕空间进行法线平滑的方式得到更为光滑的渲染结果。

2.5.3.1 PN三角形与表面细分

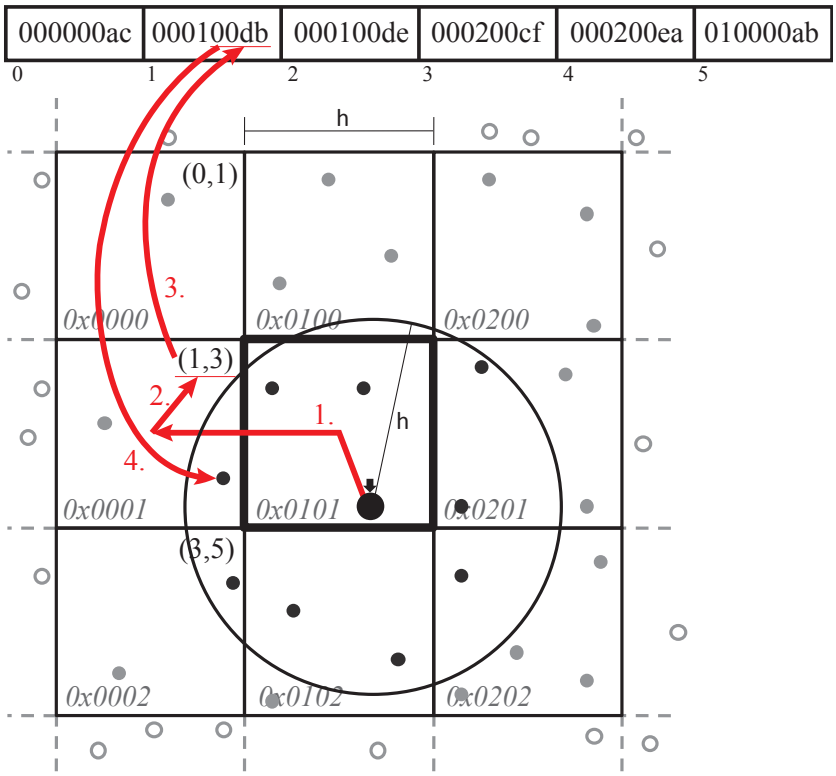

图 2.9 通过$\mathbf{b}_{300}, \mathbf{b}_{0}03, \mathbf{b}_{0}30$ 等生成平滑的细分Bezier三角形网格，图片取自参考文献[48]。

弯曲点法线三角形（PN三角形）[48]是一种用于平滑三角形网格的方法，它的特点在于非常简单高效，并且可以无缝地插入现有的图形管线中。仅仅根据三角形的三个顶点及其法线，这种技术就可以生成平滑的细分三角形。

给定三维空间中一个三角形的三个点$\vec{x}_1, \vec{x}_2, \vec{x}_3$，及其法线$\vec{n}_1, \vec{n}_2, \vec{n}_3$，首先计算出其对应的Bezier曲面控制点$\mathbf{b}_{ijk}$：

1. 将$\mathbf{b}_{ijk}$均匀分布在三角形上，三角形的三个顶点对应三个控制点$\mathbf{b}_{300}, \mathbf{b}_{030}, \mathbf{b}_{003}$。
2. 在每个角，将其对应的两条边上的最近的点投影到与法线垂直的平面上，称为切点（tangent point），如在$\mathbf{b}_{300}$ 可以生成两个切点$\mathbf{b}_{201}, \mathbf{b}_{210}$。以此类推在另两个角生成$\mathbf{b}_{102}, \mathbf{b}_{012}$ 以及$\mathbf{b}_{021}, \mathbf{b}_{120}$。
3. 将中心点朝六个切点的平均点移动，并越过移动的方向再移动1/2。

用公式总结如下：

$$
\begin{aligned}
\mathbf{b}_{300} &= \vec{x}_1, \\
\mathbf{b}_{030} &= \vec{x}_2, \\
\mathbf{b}_{003} &= \vec{x}_3, \\
\omega_{ij} &= (\vec{x}_i - \vec{x}_j) \cdot \vec{n}, \\
\mathbf{b}_{210} &= (2\vec{x}_1 + \vec{x}_2 - \omega_{12}\vec{n}_1)/3, \\
\mathbf{b}_{120} &= (2\vec{x}_2 + \vec{x}_1 - \omega_{21}\vec{n}_2)/3, \\
\mathbf{b}_{021} &= (2\vec{x}_2 + \vec{x}_3 - \omega_{23}\vec{n}_2)/3, \\
\mathbf{b}_{012} &= (2\vec{x}_3 + \vec{x}_2 - \omega_{32}\vec{n}_3)/3, \\
\mathbf{b}_{102} &= (2\vec{x}_3 + \vec{x}_1 - \omega_{31}\vec{n}_3)/3, \\
\mathbf{b}_{201} &= (2\vec{x}_1 + \vec{x}_3 - \omega_{13}\vec{n}_1)/3, \\
\mathbf{E} &= (\mathbf{b}_{210} + \mathbf{b}_{120} + \mathbf{b}_{021} + \mathbf{b}_{012} + \mathbf{b}_{102} + \mathbf{b}_{201})/6, \\
\mathbf{V} &= (\vec{x}_1 + \vec{x}_2 + \vec{x}_3)/3, \\
\mathbf{b}_{111} &= \mathbf{E} + (\mathbf{E} - \mathbf{V})/2
\end{aligned}
\tag{2-52}
$$

其中法线的控制点计算也类似，具体公式及推导过程请见参考文献[48]，这里不再赘述。

2.5.3.2 双边法线平滑

PN三角形可以解决网格过于粗糙的问题，然而，为了生成光滑的表面，仅仅用PN三角形是不够的。观察图2.11左下可以发现，PN三角形细分出的法线导致照明呈凹凸状，这主要是由于立方体匹配生成的网格中有大量的窄条，而在被PN 三角形细分之后，这些窄条形的三角形会生成更多窄条形的三角形，从而使网格质量显著下降（见图 2.10）。由于法线插值与三角形位置相关，因此

低质量的网格最终会导致法线不平整。一般来说，为了提高网格质量，可以对模型进行重网格化（re-meshing），然而这类操作通常非常耗时，不适用于实时应用。

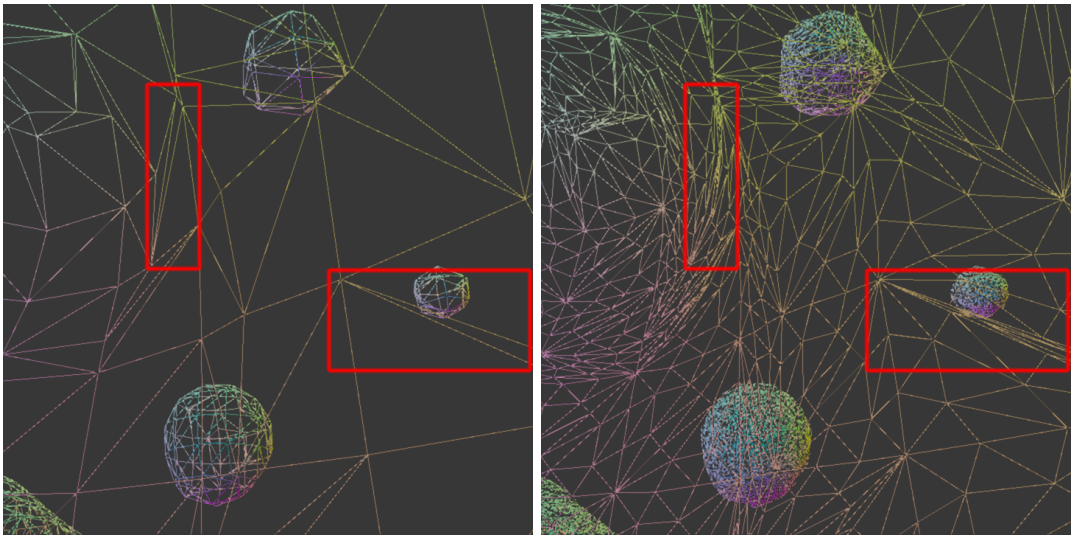

图 2.10　左：使用PN三角形细分前的网格；右：使用PN三角形细分后的网格。观察红框内区域，可以发现由于PN三角形方法的本地性，它会无法避免地在窄条形的三角形中生成更多的窄条形三角形。

由于法线平滑与否主要影响屏幕绘制时的渲染效果，本文选择在屏幕空间对法线进行双边滤波[49,50]的方法来平滑法线。这样做的优势主要有：速度快，二维双边滤波可以使用两个分离的易于并行的一维滤波器近似；效果好，实验证明这种方法可以很好地去除光滑表面法线不平整的瑕疵（见图 2.11右下）。

首先将细分的网格光栅化至两张纹理中，分别以浮点向量存储位置以及法线信息。然后使用本文设计的双边滤波器对这两张纹理进行处理。本文定义它的形式为：

$$\vec{n}_{x_{smoothed}} = \frac{1}{S(\vec{x},\vec{n}_x)}\int_{\Omega} f(\vec{x},\vec{y})g(\vec{n}_x,\vec{n}_y)\delta(\vec{y})\vec{n}_y dy \tag{2-53}$$

其中Ω是像素的邻域，

$$S(\vec{x},\vec{n}_x) = \int_{\Omega} f(\vec{x},\vec{y})g(\vec{n}_x,\vec{n}_y)\delta(\vec{y})dy \tag{2-54}$$

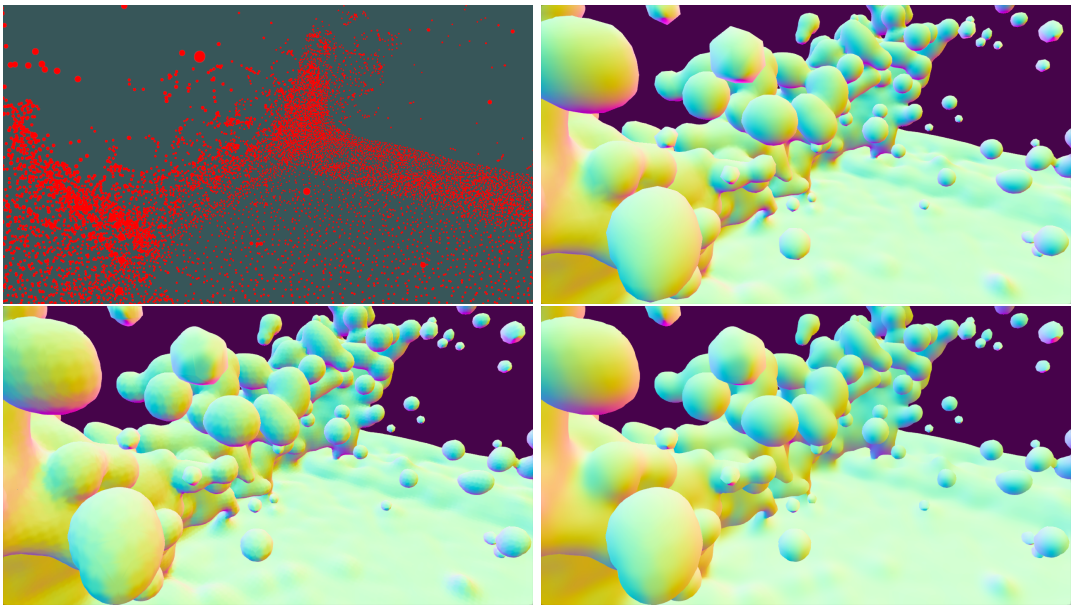

图 2.11　左上：SPH粒子分布；右上：立方体匹配重建，颜色是法线的线性映射$\frac{1}{2}(x,y,z)+\frac{1}{2}\rightarrow(R,G,B)$，可以注意网格边缘的棱角分明；左下：仅使用PN 三角形的平滑效果，可以注意到虽然网格边缘更圆滑了，然而表面法线呈凹凸状；右下：同时使用PN三角形以及双边法线平滑的效果。

以及

$$\delta(\vec{x})=\begin{cases}1 & \vec{x}\in surface\\ 0 & otherwise\end{cases} \tag{2-55}$$

该滤波器综合了空间位置以及法向量自身的信息对法向量进行平滑。它的设计概念主要源自两点：

1. 由于观察到流体的表面曲率通常不高，几乎不会出现直角和锐角，因此像素的法向相似的更可能和属于同一表面，应允许平滑，不相似的更可能属于不同表面，不应该平滑。
2. 像素的空间位置相近的更可能在流体表面上测地距离相近，因此应允许平滑；空间位置相离较远的像素其测地距离通常更远，不应该平滑。

因此，本文定义衡量空间位置的函数为高斯函数的形式：

$$f(\vec{x},\vec{y})=e^{-\frac{(\vec{x}-\vec{y})^2}{\sigma^2}} \tag{2-56}$$

同时定义法线相似度函数为：

$$g(\vec{n}_x,\vec{n}_y)=saturate(\vec{n}_x\cdot\vec{n}_y) \tag{2-57}$$

其中$saturate(x)$为将x截断到$[0,1]$之间的截断函数。

同时，为了提高效率，本文用两个分别横向和纵向的一维滤波器来代替上面的二维滤波器，即$\int_{\Omega} \approx \int_{X}\int_{Y}$。在实现中，本文设$\sigma = h$，即与SPH 的平滑核半径相等；平滑窗口$\|X\| = \|Y\| = 31$。

2.6 实现管线

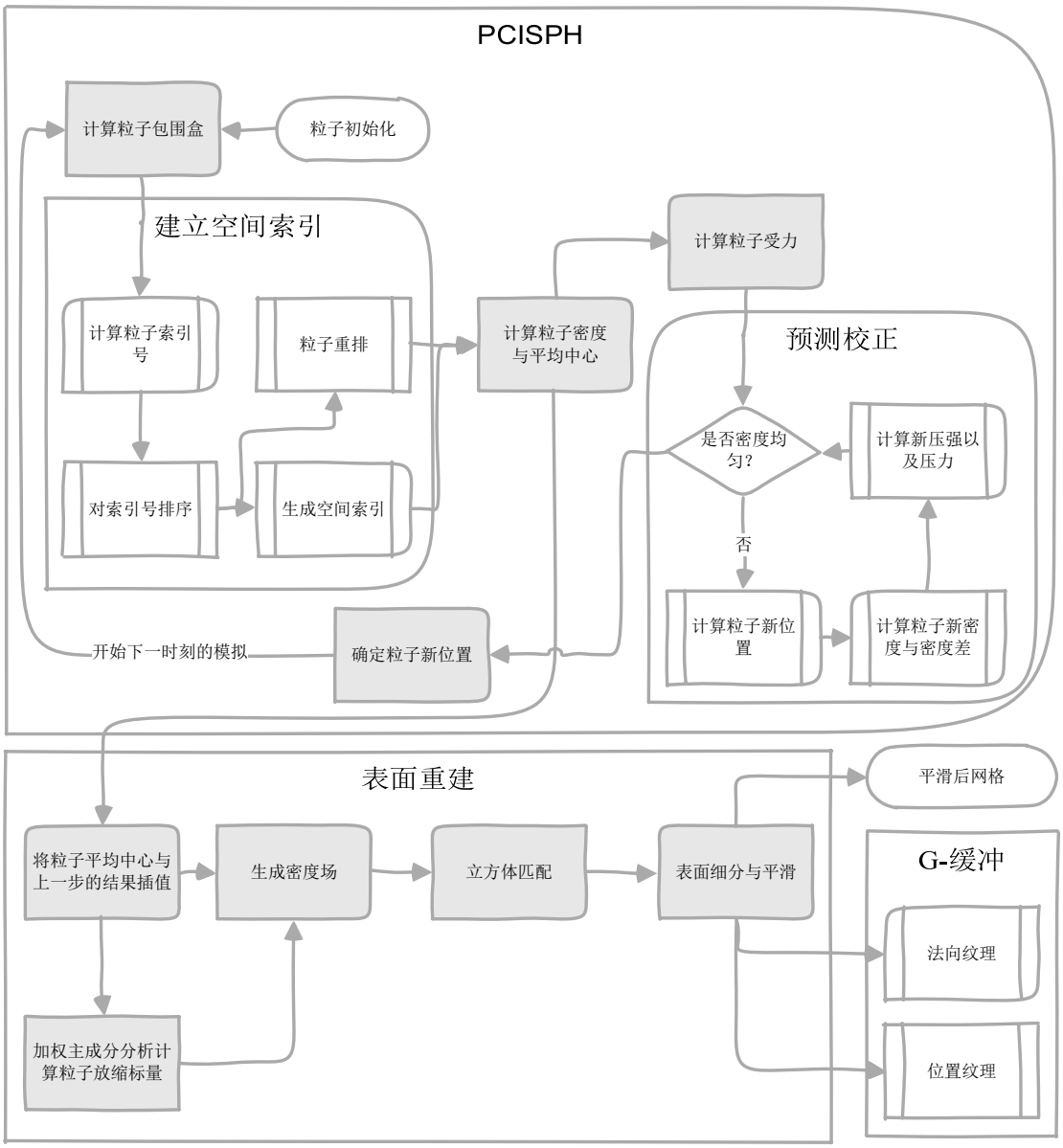

图 2.12　PCISPH与表面重建全流程。

本文的方法适用于各种并行机实现，其流程见见图 2.12。由于在个人计算机上最强大的并行计算设备就是GPU，因此本文以GPU作为实现设备，以微软Direct3D 11作为实现平台。这一节对GPU架构以及Direct3D 11的渲染管线进行介绍，并且详述本文的实现细节。

2.6.1 GPU架构

图 2.13 GPU结构示意图。

首先对GPU进行一些简介。GPU是一种特制的用于快速绘制以及显示的超大规模集成电路。通常用于图形显示以及高性能计算，事实上，现代GPU具有强大的计算能力，一般认为，一台个人计算机上的GPU的浮点运算能力等同于十年前的一台超级计算机。

GPU如此强大的计算能力来自于其高度的并行性。如图 2.13 所示，以NVIDIA的GK110系列GPU为例，一个GPU芯片中包含多个图形处理器群（Graphics Processing Clusters，或简称GPCs），每个GPC中又包含多个流处理器（Stream-Multiprocessors，或简称SMX），每个SMX中还包含多个逻辑处理器核（CUDA cores）。这些核可以同时进行工作，并行地执行同一套代码，但处理不同的数据。因此GPU正是单指令多数据（Single Instruction, Multiple Data，或SIMD）架构的典范。另外，由于GPU上线程是分组的，每组线程不仅可以通过共享片上存储器（shared memory）共用数据，线程之间切换也是零延

迟的，因此通常在考虑每个SMX可以同时处理多少个线程的时候，都把可以零延迟切换的线程也算进去，这些所有的线程称作居留线程（resident thread）。即一片GPU上可以同时运行的线程数为：

$$Threads_{GPU} = N_{SMX} * N_{threads_{resident}} \tag{2-58}$$

比如，GeForce GTX Titan拥有14个SMX，每个SMX按照计算能力3.0 规范[51]都可以同时运行2048条居留线程，也就是说，整个GPU 上可以以1006MHz同时运行$14 * 2048 = 28672$ 条线程，这远超现有的CPU的同时处理的线程数（如Intel Xeon 的某些型号最多可以同时处理16条线程）因此，对于适合并行计算的算法，GPU的效率可以高出同等多核CPU的几百倍甚至上千倍。

2.6.2 Direct3D渲染管线

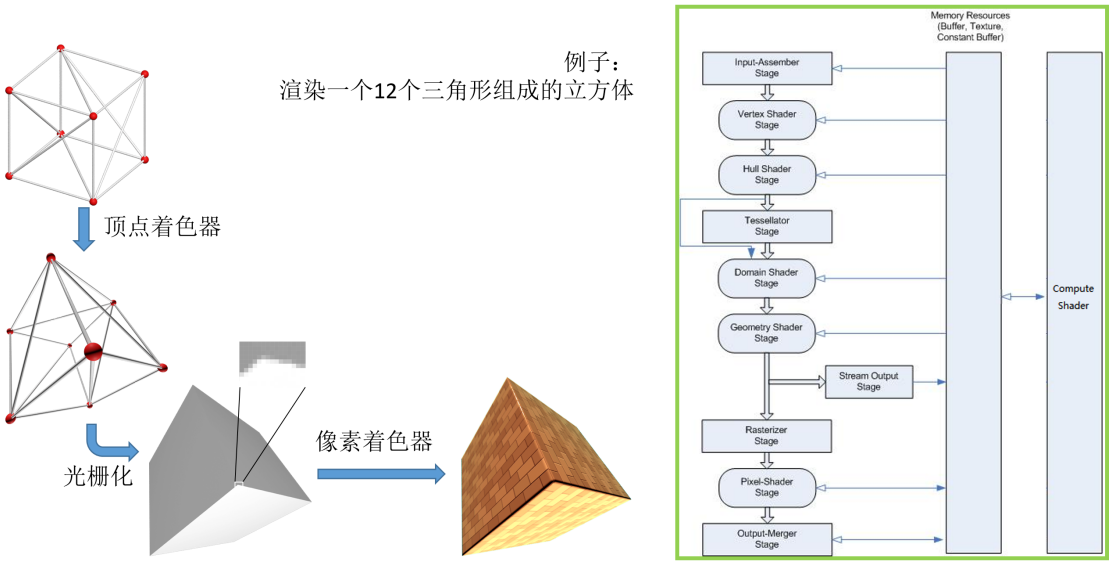

图 2.14　Direct3D渲染管线。右图更改自DirectX SDK文档。

复杂的图形编程显然不能靠直接与硬件交互完成。为了开发的简易性和规范性，图形编程中的代码要遵循一套制定好的渲染管线来运行。同时，也需要一些特定的抽象层接口（API）来与显卡驱动进行有效交互，这样的API包括用于图形绘制的Direct3D，OpenGL等，以及用于通用计算的CUDA和OpenCL。由于本文实现主要使用Direct3D 11 完成，因此在本节对Direct3D 11及其渲染管线进行简要介绍。

Direct3D 11的渲染管线分为以下几个部分（见图 2.14）：

1. **输入装配** 这个模块负责为后续的阶段提供数据，如三角形、线段、点等，在Direct3D 11中，这个阶段是可选的。
2. **顶点着色** 这个模块负责处理顶点，比如变换、蒙皮、光照等，对于一个输入的顶点，顶点着色器总是输出相应的一个顶点。
3. **壳着色** 这个模块对面片进行操作。它将输入低阶表面的控制点（如三角形的三个顶点和法线）转化为能生成一个面片的控制点（如PN三角形中的十个Bezier控制点），同时对每个面片计算一些用于镶嵌阶段以及域着色阶段的信息。
4. **镶嵌器** 这个模块会根据输入的镶嵌因子（Tessellation Factor）把输入的面片细分为更细小的面片。其具体的操作原理见参考文献 **??**。
5. **域着色** 这个模块会根据来自壳着色的控制点以及来自镶嵌器的细分出的点的局部坐标，计算输出的细分面片的每个顶点位置。
6. **几何着色** 这个模块处理整个图元。所谓图元，是对三角形、线段、点的概称。每个图元还可以包括邻接图源的信息。同时在几何着色器中，对于一个输入的图元，可以输出多个图元（或反之）。因此通常用来对图元进行复制、变异等等。这个阶段也是可选的。
7. **流输出** 这个模块的作用是将几何着色阶段产生的图元输出至显存中。这个阶段同样是可选的。
8. **光栅化** 这个模块负责将图元在屏幕上进行裁剪以及光栅化为像素，对于每个生成的像素，发起一个线程并调用像素着色器进行着色。
9. **像素着色** 这个模块为每个光栅化后的图元的每个像素进行着色。
10. **输出混合** 这个模块负责将一个像素的不同类型输入进行混合（如像素着色器的输出、深度、模版信息等），最终决定输出到屏幕或者一块显存中的值。
11. **计算着色** 这个模块完全独立于其他的模块，它可以直接对显存某一位置的数据进行并行读写、运算等，开发者可以指定发起的线程组和每组线程的数目。这个模块用于通用计算。

2.6.3 本文实现细节

对于流体模拟中的每一步，本文都使用计算着色器进行计算。在经过计算得到新的时刻SPH粒子的位置之后，同样计算着色器对三维密度场中每个体素

的密度值进行计算。

在这之后，同样的线程数被发起进行并行的立方体匹配，并将每个形成表面的三角形输出至一个AppendBuffer中（Direct3D特有的一种显存读写形式，可以不断附加新的数据，通常专用于并行变长输出）。

然后，绕过输入装配阶段，本文实现直接在顶点着色器里根据发起的顶点的索引（ID）来确定要读取AppendBuffer 中哪一个顶点的位置和法向，把他们输入到壳着色器中进行表面细分，最终通过像素着色器输出到两张四通道、每通道16位的浮点纹理中，其中一张存储像素的空间位置信息，另一张存储法线信息（这种输出方式也称作G-缓冲）。

通过绘制一张覆盖整个纹理的四边形来调用像素着色器对整张法线纹理进行双边滤波，这个过程会重复两次，分别进行横向和纵向的滤波，最终的输出同样存储在纹理中，为下一阶段的应用中要进行的着色做准备。

第 3 章　SPH方法的应用：水的流动模拟与中国糖画生成

本章引入一些实时渲染和物理交互的技术，通过两个应用从不同的侧面来展示本文的工作：水的流动模拟，以及中国糖画的交互生成。

3.1　水的流动模拟

水是生活中最常见的流体之一。它的视觉特点主要在于表面的反射与折射效果，以及由于反射或折射生成的焦散效果。本节介绍如何实时生成反射与折射。尤其介绍如何实时、同时近似无偏地生成透过水面看到的扭曲的焦散效果（LSDSE光径）。

3.1.1　菲涅耳反射与折射

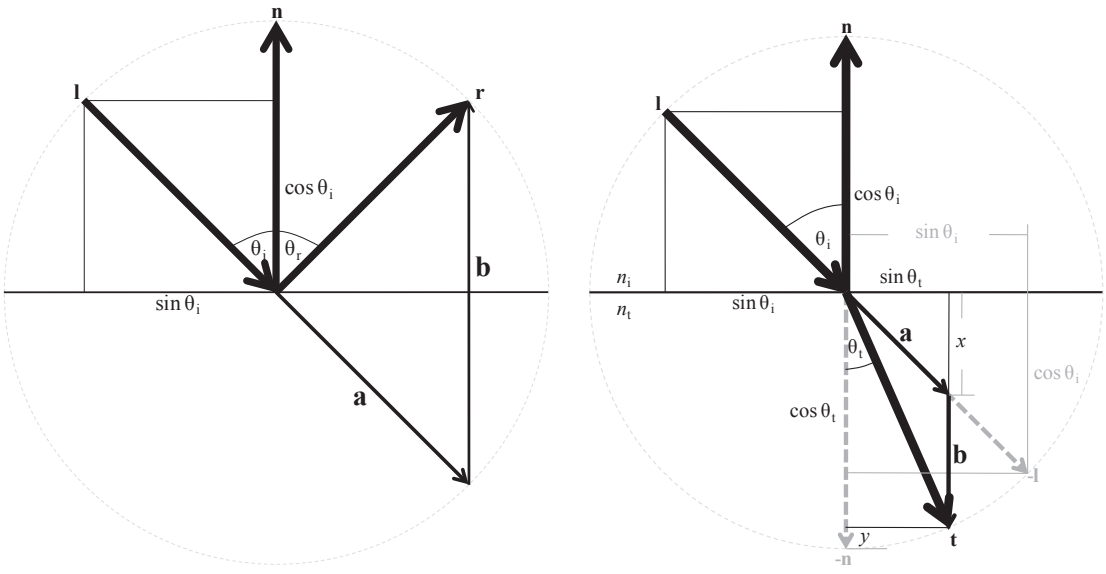

图 3.1　反射与折射定律，图片取自参考文献[38]。

首先介绍菲涅耳反射定律。如图 3.1左边所示，反射定律是非常简单的，对于给定的入射向量$\vec{l}$以及法线$\vec{n}$，其反射向量$\vec{r}$计算公式为：

$$\vec{r} = \vec{l} - 2(\vec{n} \cdot \vec{l})\vec{n} \tag{3-1}$$

其次是折射的计算。这个较反射更复杂一些。首先，折射光线与法线夹角θ_t 与入射光线与法线夹角θ_i的关系可以使用斯涅耳定律来描述，即：

$$\frac{sin\theta_i}{sin\theta_t} = \frac{n_t}{n_i} \tag{3-2}$$

其中n_t与n_i分别是折射光线一方和入射光线一方的物质的折射系数。在忽略色散以及偏振的情况下，折射方向可以通过下式计算[38]：

$$\vec{t} = \frac{n_i}{n_t}\vec{l} - (\frac{n_i}{n_t}(\vec{n}\cdot\vec{l}) + \sqrt{1-(\frac{n_i}{n_t})^2(1-(\vec{n}\cdot\vec{l})^2)})\vec{n} \tag{3-3}$$

在渲染水的时候，由于既需要处理反射也需要处理折射，因此还需要知道对于一道入射光来说，有多少分量将被反射，以及有多少分量将被折射。这可以通过菲涅耳方程来描述。对于全振光来说，菲涅耳方程可以写成以下的形式[52]：

$$R = \frac{sin^2(\theta_i-\theta_t)}{2sin^2(\theta_i+\theta_t)}(1+\frac{cos^2(\theta_i+\theta_t)}{cos^2(\theta_i-\theta_t)}) \tag{3-4}$$

其中：

$$R_0 = \frac{(n_i-n_t)^2}{(n_i+n_t)^2} \tag{3-5}$$

是入射光线垂直于入射平面时的反射率。

由于计算复杂，因此式 3-4在实际工程中通常会被其近似形式代替，称为Schlick反射率公式[53]：

$$R(\theta_i) \approx R_0 + (1-R_0)(1-cos\theta_i)^5 \tag{3-6}$$

本文中对于入射角θ_i的光线的反射率即由上式计算，其折射率为$1-R(\theta_i)$，同时，对于水使用折射系数1.333。

3.1.2 基于几何的焦散生成

焦散是一种在日常生活中非常常见的效果（图 3.2），同时也是对于水的真实感影响非常大的效果。然而，正确的模拟焦散并不是一件容易的事情，尤其是在要求实时模拟的情况下。传统的生成焦散方法可以追溯到一些较早期的渲染研究，如双向路径追踪（BPT）[55]，以及梅特罗波利斯光传输（MLT）[56]等。这些方法可以生成无偏的焦散效果，但即便是此类之中最新的

图 3.2　左：阳光经水面反射在船体上生成焦散效果。右：阳光经水面折射以及盆体反射在盆底生成焦散效果。照片来自Physics around us[54]。

方法[57]，在GPU 上实现也需要很长时间（几分钟甚至几小时）才可以收敛，否则就会具有很大的随机噪声，因此它们并不适用于实时应用。另一类方法基于光子映射[58]。光子映射通常分为两步：第一步由光源发射光子，追踪光子的反射与折射，并记录其打在漫反射表面时的位置、颜色以及入射方向；第二步从摄像机发射光线，经由镜面（如水等）的反射与折射后确定其打在漫反射表面的位置，在该位置对光子进行收集、颜色统计，从而得出最终像素的颜色。这种方法可以很有效地使用GPU完全并行化[59,60]，但这些方法也继承了光子映射的缺点：要对本文所涉及的场景生成尖锐的焦散效果，即便使用一些视点自适应的策略，通常也需要上千万的光子[61]才能达到良好的效果，无法用于实时应用。

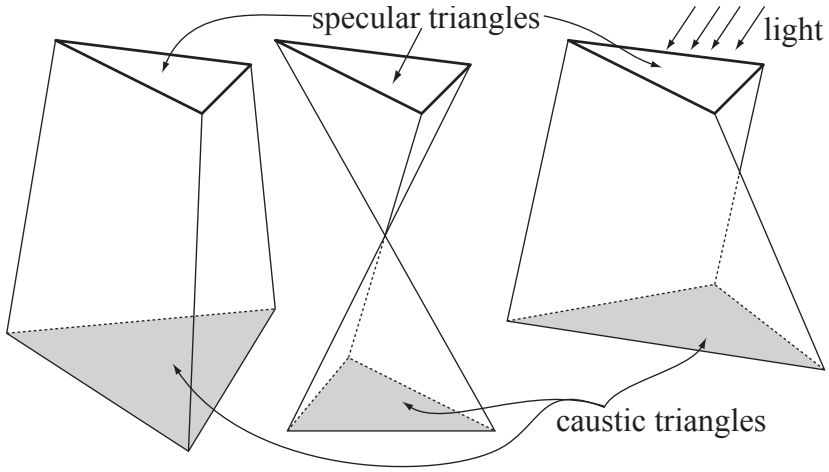

图 3.3　镜面三角形与焦散三角形。图片取自参考文献[62]。

还有另一类完全不同的方法，称为几何法[62]。这类方法在三角形的三个顶

点上根据其法线和入射光线计算出折射（或反射）方向，然后直接在三条折射（或反射）光线与投影面相交的三个点生成焦散三角形（图 3.3），并通过两个三角形面积比以及各个光线与表面的夹角计算出生成的三角形的亮度，然后将所有三角形的焦散三角形亮度叠加混合，即可生成尖锐的焦散效果。这种方法的问题在于需要三角形非常细致：首先，当折射反射次数较多时，需要对三角形不断细分才能生成正确的焦散结果；即便折射反射只有一次，也需要三角形足够细，焦散才不会显示出有强烈棱角的边缘。

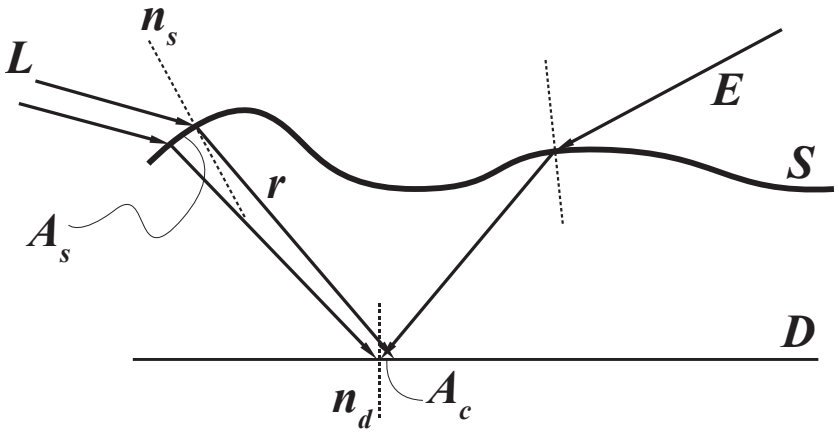

图 3.4　LSDSE路径

幸而，对于水面来说，光线通常从上方射入，同时折射反射的次数也不会太多，事实上，通常使用一次即可满足人眼对真实感的需求；另外，由于由移动立方体生成的三角形非常细致，折射反射而成的焦散三角形也会非常细致，可以生成平滑的焦散。因此，本文中使用几何法来生成焦散。另外，为了能从摄像机看到被水面扭曲的焦散效果（LSDSE路径，见图 3.4），本文结合光子映射的思路，使用纹理来存储生成的焦散效果，然后再在摄像机中绘制画面的时候在像素着色器中对纹理采样来把焦散绘制到屏幕上。

总的来说，本文的焦散渲染分为两步：

1. 首先对每个三角形的折射光线进行计算，即将从光源*L*发射的光线经过水面三个顶点的法向折射，并与漫反射的投影面*D* 求交，并在交点处生成焦散三角形，这些三角形会被叠加混合到一张焦散阴影纹理中（图 3.5，阴影使用类似方法投影生成）。
2. 然后从摄像机渲染场景，包括水面，对于水面的每个像素，计算摄像机到像素的方向*E*被水面*S*折射到*D*上的位置，然后将位置变换到焦散阴影纹

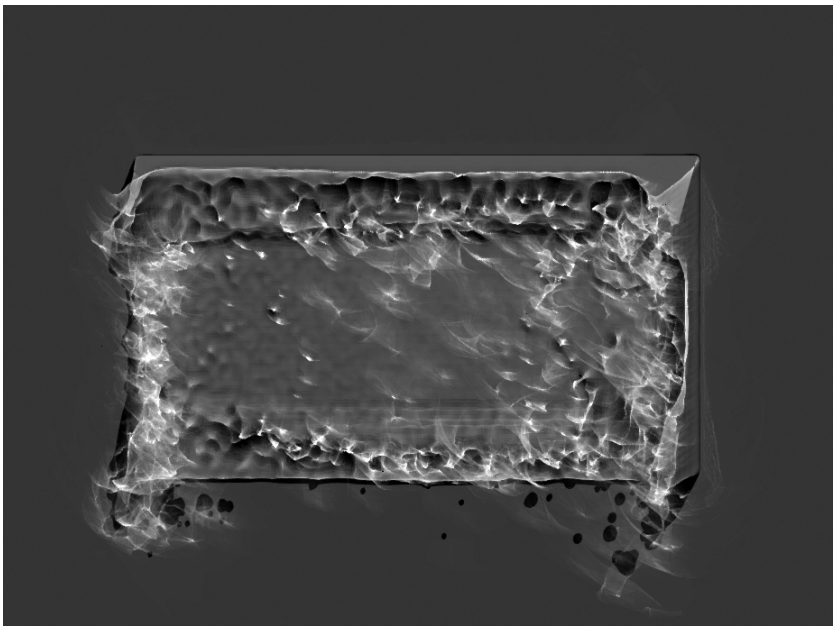

图 3.5　焦散阴影纹理。

理的坐标系中，对纹理进行采样从而得到焦散的亮度。

3.

为了简便起见，本文直接使用D的解析形式进行求交。如对于D 为高度为零的水平面的情况，设入射点为$\vec{x}$，则交点$\vec{p}$ 的计算公式为：

$$\vec{p} = \vec{x} + (\vec{x} \cdot \vec{n}_d)(\vec{r} - (\vec{r} \cdot \vec{n}_d)\vec{n}_d) \tag{3-7}$$

当然，本文的方法也支持使用图像空间求交[63]或使用Optix[64]等其他方式，从而可以支持更复杂的投影表面。

下面分析每个焦散三角形的亮度计算。首先需要考虑入射光线与镜面三角形的夹角，这和入射光的光通量有关，而显然光通量与三角形在入射光线方向的投影面积有关，因此入射光通量：

$$\Phi = A_s L_s (\vec{n}_s \cdot \vec{L}) \tag{3-8}$$

其中L_s是入射光亮度。在入射光被水面折射之后，会在投影面上生成焦散三角形，面积为A_c，考虑入射光的能量（光通量）恒定（即不随水深而衰减），因此亮度仅与三角形面积呈反比；又由于漫反射表面的亮度随入射光与法线夹角

增大而衰减，因此$L_c = \frac{\Phi}{A_c}(\vec{n}_d \cdot \vec{r})$，代入$\Phi$得到：

$$L_c = \frac{A_s L_s}{A_c}(\vec{n}_s \cdot \vec{L})(\vec{n}_d \cdot \vec{r}) \tag{3-9}$$

在实际实现中，本文在顶点着色器中进行求交计算，在几何着色器中生成折射与反射的焦散三角形；对于像素着色器输入的每个像素的亮度，使用式 3-10进行计算，然后送入显卡的硬件混合器中进行颜色叠加。

3.1.3 结果展示

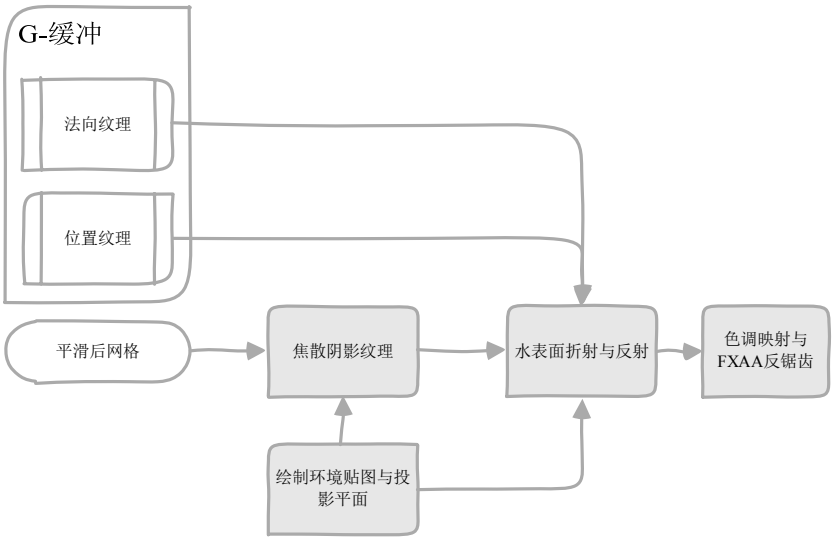

图 3.6 渲染水的流程图。

本文方法的流程见图 3.6。为了达到更好的渲染质量，本文实现将输出进行色调映射[65]以及屏幕空间反锯齿（FXAA）[66]处理。最终的模拟效果如图 3.7 所示。

3.2 中国糖画的交互生成

本节介绍本文方法的另一个应用，即一套中国糖画的交互生成系统。中国糖画是一种古老的艺术。糖画艺人使用熔化的糖液作为颜料进行绘制，最终等糖液冷却以后，即形成既有艺术性又色香味俱全的中国糖画（如图 3.8）。

本文目标是初步实现一套可交互的系统，艺术家可以通过使用该系统制造出数字化的糖画，其中熔化糖液的运动使用PCISPH来模拟。糖液的特点是温度

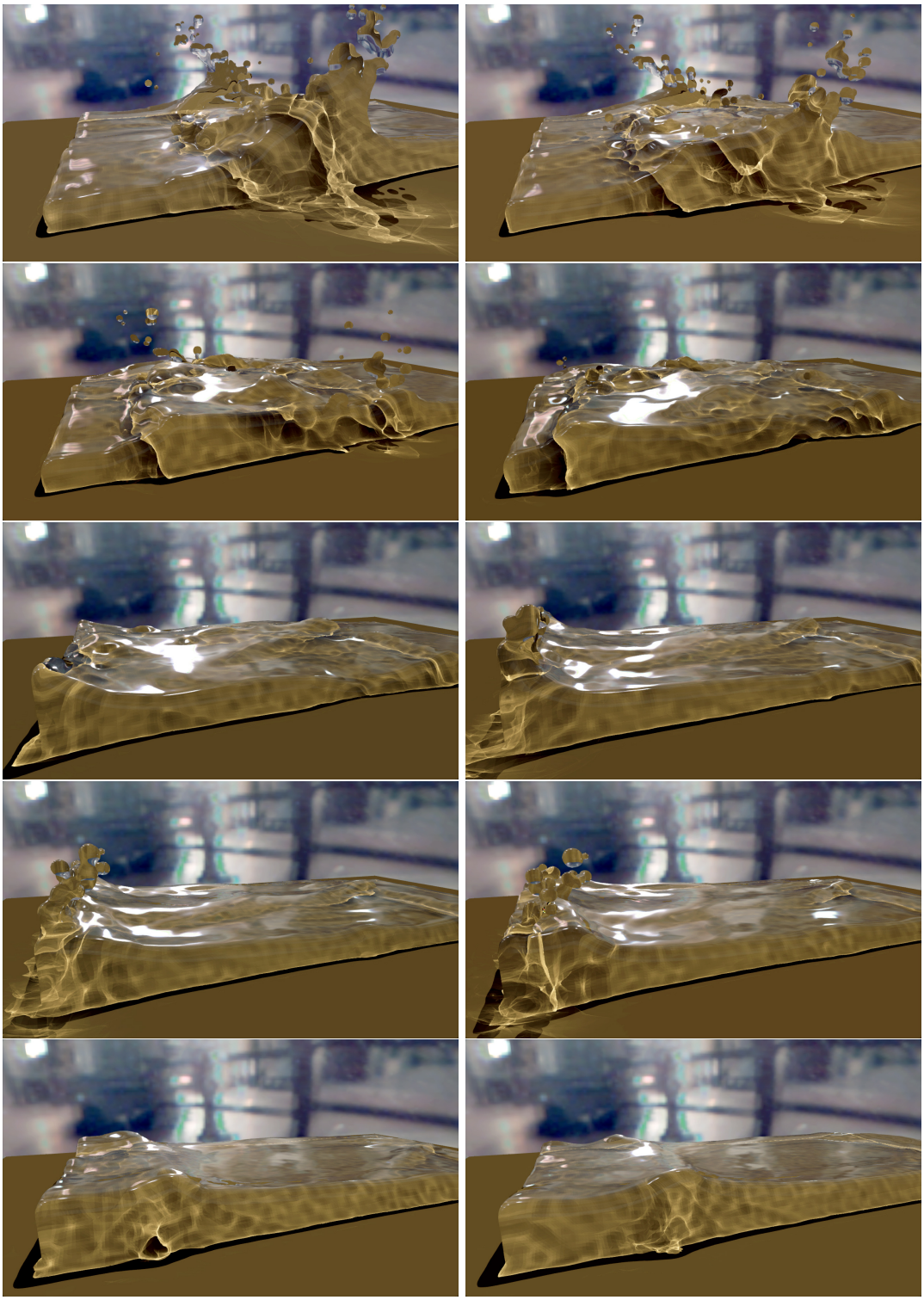

图 3.7　本文的水模拟效果。

图 3.8　现实中的中国糖画。

高粘性大，随着温度下降，粘性会逐渐加大；并且热量会从高温区扩散到低温区。因此还需要处理温度扩散以及变化粘性液体的模拟。

3.2.1　温度扩散与变粘性液体模拟

本文沿承Müller等人的方法[67]来计算温度扩散。与其他的物理量扩散一样，温度扩散也遵循一套扩散方程，如对于物理量$A(\vec{x},t)$的扩散方程为：

$$\frac{\partial A}{\partial t} = c\nabla \cdot \nabla A \tag{3-10}$$

把它写成SPH的形式，则粒子的属性A_i即可计算为：

$$\frac{\partial A_i}{\partial t} = c\sum_j \frac{A_j - A_i}{\rho_j}\nabla \cdot \nabla W(\vec{r}_{ij}, h) \tag{3-11}$$

对于每步计算的$\frac{\partial A_i}{\partial t}$采用简单的欧拉策略累加：

$$A_i(t+1) = A_i(t) + \delta t\frac{\partial A_i}{\partial t} \tag{3-12}$$

对于温度即可以上述策略计算。

由于液体粘性与温度的关系非常复杂，因此本文对一些实验数据进行拟合，并得到对温度T的如下经验公式：

$$\eta = clamp(\lambda T^{-\tau}, \eta_{min}, \eta_{max}) \tag{3-13}$$

在实现中使用$\tau = 2.17$，$\lambda = 1.01 \times 10^6$，$\eta_{min} = 0.29$，$\eta_{max} = 1.787$。

由于液体具有很高的粘性，为了尽可能真实地模拟这种效果还需要引入粒子之间的弹性力。这里本文沿用Clavet等人的方法[68]：对每一对粒子计算由于

他们之间的弹力导致的加速度为：

$$\vec{a}_{elasticity} = \frac{1}{2}k(1 - L_{ij}/h)(L_{ij} - r_{ij})\vec{r}_{ij} \tag{3-14}$$

其中L_{ij}是粒子间弹簧的固有长度。在实际应用中进行速度计算的时候使$k \sim \eta$，从而粒子之间的弹力随着温度下降而加强。

3.2.2 液体固体交互

为了使得糖液可以与糖勺以及纸张正确地交互，本文沿承虚体SPH方法[69]，将纸张以及糖勺同样离散化为SPH粒子，并将他们的密度设为与液体一样，从而避免它们被液体粒子穿透。同时，在每步计算中，将他们视为与液体粒子一样进行密度计算、压强计算等等，只是不使他们参与最终的位置移动。另外对于他们与液体之间的压力，本文实现使用惩罚力方式，即：

$$\vec{a}_{i\,penalty} = -min(\vec{x}_i \cdot \vec{n}_{obj}, 0)\kappa_{obj}\vec{n}_{obj} \tag{3-15}$$

其中κ_{obj}是物体的表面刚性，$\vec{n}_{obj}$是粒子i邻域中的物体粒子平均法向。

在虚体SPH中，固体表面上的粒子分布应服从蓝噪声分布。本文沿用Bowers等人的方法[70]在三角形网格表面进行蓝噪声点采样。本文中的实现过程如下：

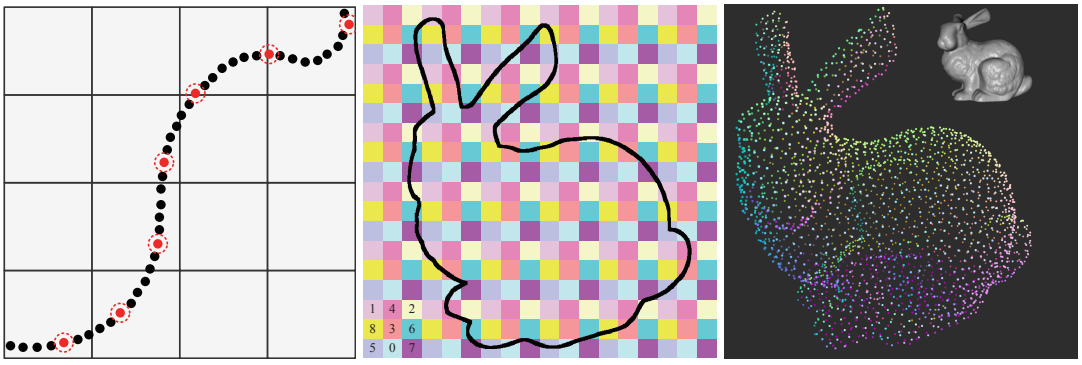

图 3.9 并行表面蓝噪声采样。左图与中图取自参考文献[70]；右图是本文实现的结果。

1. 受到之前工作[71]的启发，本文使用镶嵌器对输入的三角形网格进行表面细分，然后把输出的每个三角形用其重心代替，输出为一个候选的采样点（如图 3.9 左所示）。

2. 将候选的采样点所在的空间进行均匀划分（如图 3.9右所示），划分的格子边长为$\sqrt{3}h$。将每27个相邻格子分为27组（如图 3.9右的左下角所示，由于图示为二维，因此分为了9组）。
3. 用一个从1到27的随机序列作为循环的索引j，循环27次。在每次循环中，首在属于第j组的每个格子里随机选出一个采样点，然后使用同第 2.4节中的并行检索算法检测距它半径h的邻域内是否已经有点被选中，如果有则放弃选择这个采样点，否则将其加入选中采样点的序列中。在估算点与点之间的测地距离时，本文采用对Bowers等人工作[70]的改进方法进行计算，从而避免了其无法处理法线相同而位置向量与法线平行情况的缺陷。本文使用的公式为：

$$\begin{aligned} c_1 &= \vec{n}_1 \cdot normalize(p_2 - p_1) \\ c_2 &= \vec{n}_2 \cdot normalize(p_2 - p_1) \\ d_g &= \begin{cases} \|\vec{p}_1 - \vec{p}_2\| & c_1 = c_2 \\ \frac{ArcSin(c_2) - ArcSin(c_1)}{2(c_1 - c_2)} \sqrt{2(1 - c_1 c_2)} + (1 - \frac{\sqrt{2}}{2}) & otherwise \end{cases} \end{aligned} \tag{3-16}$$

本文的并行表面蓝噪声采样使用计算着色器实现。为了进一步提高效率本文实现了一套并行的Sobol伪随机数生成器，其具体原理及算法请见参考文献[72]，在此不再赘述。图 3.9右展示了本文方法的效果。

3.2.3 结果展示

本文使用基本的鼠标键盘作为输入方式，后续工作可以引入如Kinect等[73]进行输入，从而达到更佳的交互体验。本文的效果见图 3.10。

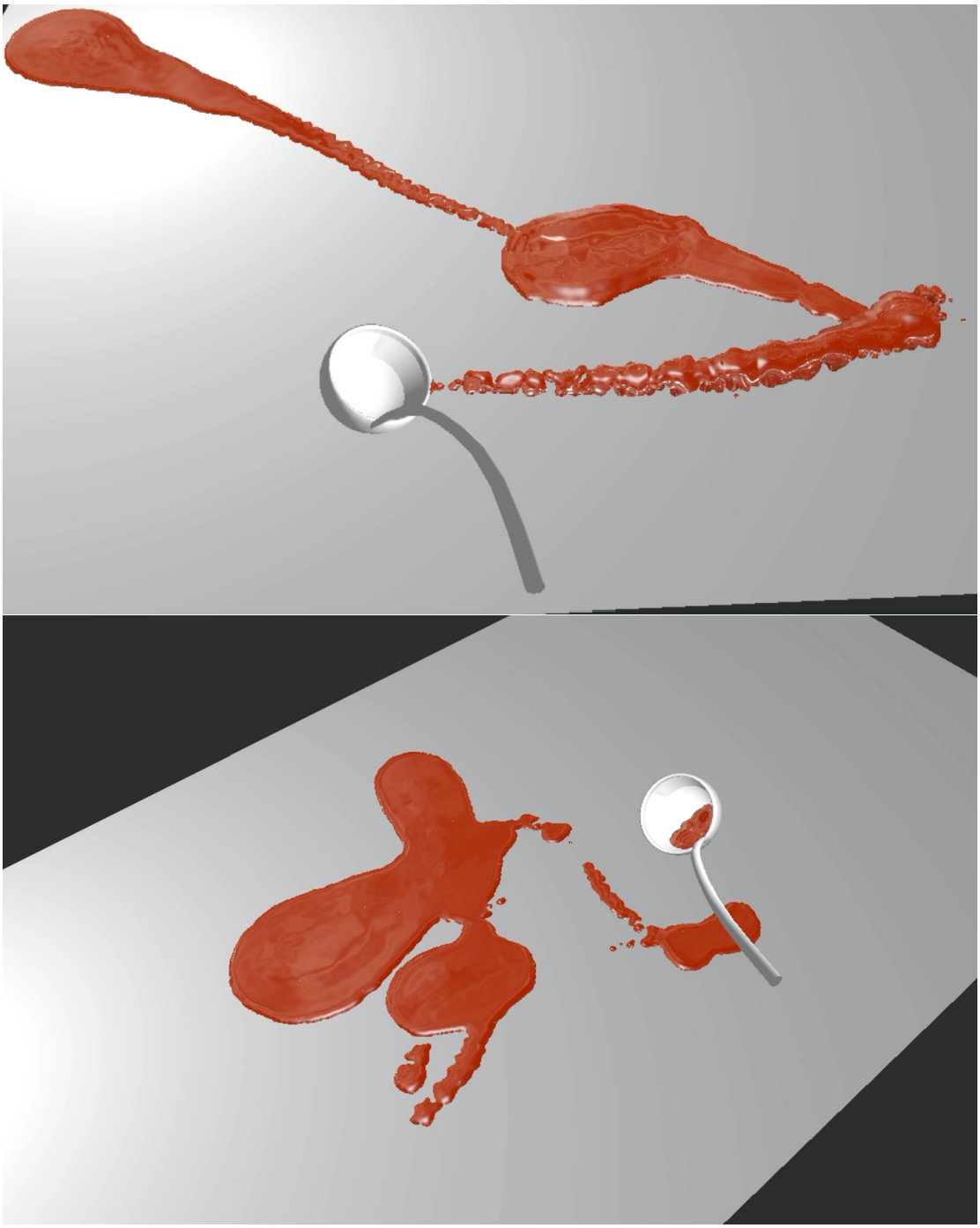

图 3.10　本文的糖画交互绘制。

第 4 章　结论与展望

本文详述了PCISPH算法的理论基础以及工程实现方法，并介绍了如何使用图形处理单元对其进行并行加速；同时，本文展示了方法的两个应用：水的模拟，以及中国糖画的交互生成。本文的开源实现（见https://www.dropbox.com/s/cqdqkg0v9cay18d/FluidCS113D.zip）可以在现代GPU上实时运行，并可以基本真实地模拟流体的各种现象。

4.1　性能

本文实验的环境为Intel i7 920处理器和NVIDIA GeForce GTX Titan。下面对其性能进行具体讨论。本文以水模拟的例子（即图 3.7中的场景）来评估本文

粒子数	8K	16K	32K	64K	128K
PCISPH	1.1	1.2	2.5	7.1	16.1
密度场生成	0.4	0.7	0.8	1.6	3.9
移动立方体	1.5	1.3	1.2	1.9	2.2
焦散与阴影	5.1	5.7	6.0	6.9	7.2
PN三角形	0.3	0.5	0.6	0.7	0.5
双边滤波	0.2	0.2	0.8	0.4	0.5
总耗时	**8.6**	**9.7**	**11.9**	**18.7**	**30.4**

表 4.1　本文工作的性能统计（以水模拟为例），计时单位为毫秒（ms）。

工作的性能，并对每个阶段分别计时（见表 4.1），计时误差为1.6ms。

从计时结果可以发现PN三角形细分与双边滤波几乎不耗任何时间，并且本文的密度场生成以及移动立方体等阶段的效率也非常高。其中最耗时的依然是PCISPH，其次是焦散与阴影的生成。可以观察到焦散及阴影生成与粒子数的多少关系并不明显：由于焦散与阴影生成的发起线程数仅仅与三角形数目相关，因此这种情况是很合理的。同时可以观察到PCISPH的耗时在GPU计算能力被充分利用时与粒子数呈线性关系（见图 4.1），注意到这种线性关系自粒子数超过GPU的居留线程数（28672，见第 2.6.1 节）时尤其明确：由于当需要处理

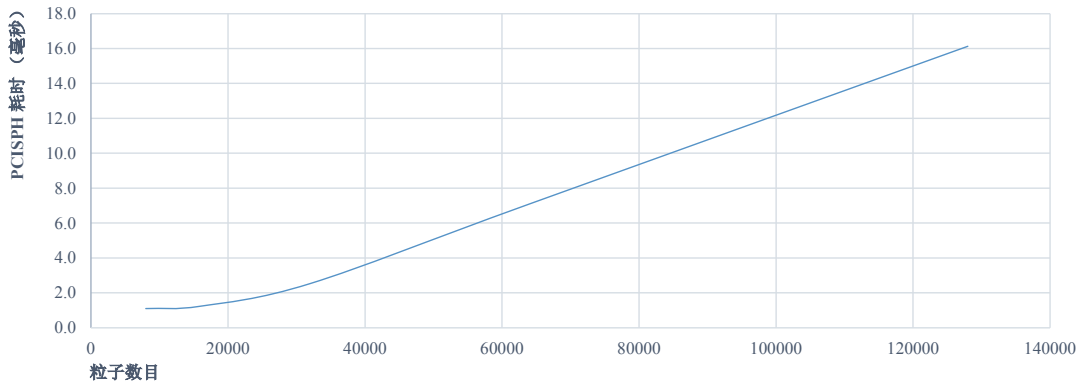

图 4.1　PCISPH的耗时/粒子数目曲线。

的粒子数大于GPU的居留线程数时，粒子便会被分组，并且不同的组之间会被随机但串行地处理，因此这种情况也是不难理解的。

4.2　未来工作

由于本文的工作基于PCISPH完成，因此仍然沿承了它的一些缺点。首先，本文的最近邻搜索的效率仍有待提高，由于SPH算法要求精确的最近邻搜索，这使得在近似最近邻（Approximated Nearest Neighbor, ANN）搜索上的进展（如Li等人的工作[22]）难以引入。因此探索近似最近邻搜索在粒子流体模拟上的应用将是一个有潜力的研究方向；其次，由于PCISPH对保证模拟稳定性的时间步长限制较为严格，因此如何提升粒子流体模拟的时间步长也是一个亟待解决的问题，最近Macklin等人的工作[74]在这方面取得了重大进展，然而要满足业界的需求（如游戏开发等）依然需要在这一方面投入更多的研究。

另外在本文提出的两个应用中，也有许多亟待改进的地方。如在水的模拟中，由于需要生成焦散，因此必须使用移动立方体生成网格，而实际上很多无网格[20,32–35]（或部分网格[31]）的表面重建方法的效率远高于此方法。因此探索无网格情况下水的真实渲染也是一个有潜力的研究方向。对于中国糖画模拟，本文对于粘性力模拟仍然比较初级，引入更精巧的算法[75]预计可以取得更佳的效果，事实上，使用不可压NS方程来模拟熔化糖液的合理性也需要更多的探讨与证明，如需要与在考虑糖液可压的情况下的离线模拟[76]做更多的比较与分析。

插图索引

表格索引

参考文献

[1] Solenthaler B, Pajarola R. Predictive-corrective incompressible SPH. ACM Transactions on Graphics (TOG), 2009, 28(3):1

[2] Zhu Y, Bridson R. Animating sand as a fluid. ACM Transactions on Graphics (TOG), 2005, 24(3):965–972

[3] Akinci G, Ihmsen M, Akinci N, et al. Parallel Surface Reconstruction for Particle-Based Fluids. Computer Graphics Forum (CGF), 2012, 31(6):1797–1809

[4] Gingold R A, Monaghan J J. Smoothed particle hydrodynamics-theory and application to non-spherical stars. Monthly notices of the royal astronomical society, 1977, 181:375–389

[5] Lucy L B. A numerical approach to the testing of the fission hypothesis. The astronomical journal, 1977, 82:1013–1024

[6] Müller M, Charypar D, Gross M. Particle-based fluid simulation for interactive applications. Proceedings of ACM SIGGRAPH/Eurographics symposium on Computer animation. Eurographics Association, 2003. 154–159

[7] Batchelor G K. An introduction to fluid dynamics. Cambridge university press, 2000

[8] Becker M, Teschner M. Weakly compressible SPH for free surface flows. Proceedings of ACM SIGGRAPH/Eurographics symposium on Computer animation. Eurographics Association, 2007. 209–217

[9] Ihmsen M, Akinci N, Gissler M, et al. Boundary Handling and Adaptive Time-stepping for PCISPH. Proceedings of Workshop in Virtual Reality Interactions and Physical Simulation. The Eurographics Association, 2010. 79–88

[10] Kass M, Miller G. Rapid, stable fluid dynamics for computer graphics. ACM SIGGRAPH Computer Graphics, 1990, 24(4):49–57

[11] Tessendorf J, et al. Simulating ocean water. ACM SIGGRAPH Courses, 2001.

[12] Maes M M, Fujimoto T, Chiba N. Efficient animation of water flow on irregular terrains. Proceedings of International conference on Computer graphics and interactive techniques in Australasia and Southeast Asia. ACM, 2006. 107–115

[13] Yuksel C, House D H, Keyser J. Wave particles. ACM Transactions on Graphics (TOG), 2007, 26(3):99

[14] Št'ava O, Beneš B, Brisbin M, et al. Interactive terrain modeling using hydraulic erosion. Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation. Eurographics Association, 2008. 201–210

[15] Amada T, Imura M, Yasumuro Y, et al. Particle-based fluid simulation on gpu. Proceedings of ACM Workshop on General-Purpose Computing on Graphics Processors and SIGGRAPH, 2004

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

[17] Zhang Y, Solenthaler B, Pajarola R. Adaptive sampling and rendering of fluids on the GPU. Proceedings of Eurographics/IEEE VGTC conference on Point-Based Graphics. Eurographics Association, 2008. 137–146

[18] Le Grand S. Broad-phase collision detection with CUDA. GPU Gems, 2007, 3:697–721

[19] Microsoft. FluidCS11. Direct3D SDK, 2010.

[20] Goswami P, Schlegel P, Solenthaler B, et al. Interactive SPH simulation and rendering on the GPU. Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation. Eurographics Association, 2010. 55–64

[21] Morton G M. A computer oriented geodetic data base and a new technique in file sequencing. International Business Machines Company, 1966

[22] Li S, Simons L C, Pakaravoor J B, et al. kANN on the GPU with Shifted Sorting. Proceedings of High Performance Graphics, 2012

[23] Lorensen W E, Cline H E. Marching cubes: A high resolution 3D surface construction algorithm. Proceedings of ACM Siggraph Computer Graphics, volume 21. ACM, 1987. 163–169

[24] Blinn J F. A generalization of algebraic surface drawing. ACM Transactions on Graphics (TOG), 1982, 1(3):235–256

[25] Yu J, Turk G. Reconstructing surfaces of particle-based fluids using anisotropic kernels. ACM Transactions on Graphics (TOG), 2013, 32(1):5

[26] Solenthaler B, Schläfli J, Pajarola R. A unified particle model for fluid–solid interactions. Computer Animation and Virtual Worlds (CAVW), 2007, 18(1):69–82

[27] Müller M. Fast and robust tracking of fluid surfaces. Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation. ACM, 2009. 237–245

[28] Brochu T, Bridson R. Robust topological operations for dynamic explicit surfaces. SIAM Journal on Scientific Computing, 2009, 31(4):2472–2493

[29] Wojtan C, Thürey N, Gross M, et al. Physics-inspired topology changes for thin fluid features. ACM Transactions on Graphics (TOG), 2010, 29(4):50

[30] Yu J, Wojtan C, Turk G, et al. Explicit mesh surfaces for particle based fluids. Computer Graphics Forum (CGF), 2012, 31(2pt4):815–824

[31] Müller M, Schirm S, Duthaler S. Screen space meshes. Proceedings of ACM SIGGRAPH/Eurographics symposium on Computer animation. Eurographics Association, 2007. 9–15

[32] Fraedrich R, Auer S, Westermann R. Efficient high-quality volume rendering of SPH data. IEEE Transactions on Visualization and Computer Graphics (TVCG), 2010, 16(6):1533–1540

[33] Zwicker M, Pfister H, Van Baar J, et al. Surface splatting. Proceedings of Annual conference on Computer graphics and interactive techniques. ACM, 2001. 371–378

[34] Adams B, Lenaerts T, Dutré P. Particle splatting: Interactive rendering of particle-based simulation data. CW Reports, 2006. 16

[35] Laan W J, Green S, Sainz M. Screen space fluid rendering with curvature flow. Proceedings of Symposium on Interactive 3D graphics and games. ACM, 2009. 91–98

[36] Bridson R. Fluid Simulation. SIGGRAPH 2007 Course Notes, 2007. 1–81

[37] White F M. Viscous Fluid Flow 3e. Tata McGraw-Hill Education, 1974

[38] Auer S. Realtime particle-based fluid simulation

[39] Blelloch G E. Vector models for data-parallel computing, volume 75. MIT press Cambridge, 1990

[40] Microsoft. BitonicSortCS11. Direct3D SDK, 2010.

[41] Harris M, Sengupta S, Owens J D. Parallel prefix sum (scan) with CUDA. GPU gems, 2007, 3(39):851–876

[42] Satish N, Harris M, Garland M. Designing efficient sorting algorithms for manycore GPUs. Proceedings of IEEE International Symposium on Parallel & Distributed Processing. IEEE, 2009. 1–10

[43] Bell N, Hoberock J. Thrust: A productivity-oriented library for CUDA. GPU Computing Gems: Jade Edition, 2011. 359–372

[44] Monaghan J. On the problem of penetration in particle methods. Journal of Computational physics, 1989, 82(1):1–15

[45] Koren Y, Carmel L. Visualization of labeled data using linear transformations. Proceedings of Information Visualization, volume 2003. IEEE, 2003

[46] Kopp J. Efficient numerical diagonalization of hermitian 3× 3 matrices. International Journal of Modern Physics C, 2008, 19(03):523–548

[47] Knapp C E. An implicit smooth particle hydrodynamic code. Technical report, Los Alamos National Lab., NM (US), 2000

[48] Vlachos A, Peters J, Boyd C, et al. Curved PN triangles. Proceedings of Symposium on Interactive 3D graphics. ACM, 2001. 159–166

[49] Tomasi C, Manduchi R. Bilateral filtering for gray and color images. Proceedings of International Conference on Computer Vision. IEEE, 1998. 839–846

[50] He K, Sun J, Tang X. Guided image filtering. Proceedings of Computer Vision–ECCV 2010. Springer, 2010: 1–14

[51] Corporation N. NVIDIA CUDA programming guide, 2011

[52] Wloka M. Fresnel reflection technical report. Technical report, NVIDIA Corporation, 2002

[53] Schlick C. A customizable reflectance model for everyday rendering. Proceedings of Eurographics Workshop on Rendering. Paris, France, 1993. 73–83

[54] Pieranski P. Physics Around Us. http://etacar.put.poznan.pl/piotr.pieranski/Physics%20Around%20Us/Physics%20around%20us.html

[55] Lafortune E P, Willems Y D. Bi-directional path tracing. Proceedings of CompuGraphics, volume 93, 1993. 145–153

[56] Veach E, Guibas L J. Metropolis light transport. Proceedings of Annual conference on Computer graphics and interactive techniques. ACM Press/Addison-Wesley Publishing Co., 1997. 65–76

[57] Kaplanyan A S, Dachsbacher C. Path Space Regularization for Holistic and Robust Light Transport. Computer Graphics Forum (CGF), 2013, 32(2)

[58] Jensen H W. Realistic image synthesis using photon mapping. AK Peters, Ltd., 2001

[59] Yao C, Wang B, Chan B, et al. Multi-Image Based Photon Tracing for Interactive Global Illumination of Dynamic Scenes. Computer Graphics Forum (CGF), 2010, 29(4):1315–1324

[60] Fei Y, Wang B. Fast multi-image-based photon tracing with grid-based gathering. Proceedings of ACM SIGGRAPH 2012 Posters. ACM, 2012. 105

[61] Kaplanyan A S, Dachsbacher C. Adaptive progressive photon mapping. ACM Transactions on Graphics (TOG), 2013, 32(2):16

[62] Ernst M, Akenine-Möller T, Jensen H W. Interactive rendering of caustics using interpolated warped volumes. Proceedings of Graphics Interface. Canadian Human-Computer Communications Society, 2005. 87–96

[63] Szirmay-Kalos L, Aszódi B, Lazányi I, et al. Approximate Ray-Tracing on the GPU with Distance Impostors. Proceedings of Computer Graphics Forum, volume 24. Wiley Online Library, 2005. 695–704

[64] Parker S G, Bigler J, Dietrich A, et al. Optix: a general purpose ray tracing engine. ACM Transactions on Graphics (TOG), 2010, 29(4):66

[65] Duiker H P. Filmic Tonemapping for Real-time Rendering. Proceedings of ACM SIGGRAPH Courses. ACM, 2010

[66] Jimenez J, Gutierrez D, Yang J, et al. Filtering approaches for real-time anti-aliasing. ACM SIGGRAPH Courses, 2011, 2(3):4

[67] Müller M, Solenthaler B, Keiser R, et al. Particle-based fluid-fluid interaction. Proceedings of ACM SIGGRAPH/Eurographics symposium on Computer animation. ACM, 2005. 237–244

[68] Clavet S, Beaudoin P, Poulin P. Particle-based viscoelastic fluid simulation. Proceedings of ACM SIGGRAPH/Eurographics symposium on Computer animation. ACM, 2005. 219–228

[69] Schechter H, Bridson R. Ghost sph for animating water. ACM Transactions on Graphics (TOG), 2012, 31(4):61

[70] Bowers J, Wang R, Wei L Y, et al. Parallel Poisson disk sampling with spectrum analysis on surfaces. ACM Transactions on Graphics (TOG), 2010, 29(6):166

[71] Fei Y, Wang B, Chen J. Point-tessellated voxelization. Proceedings of Graphics Interface. Canadian Human-Computer Communications Society, 2012. 9–18

[72] Bradley T, Toit J, Giles M, et al. Parallelisation Techniques for Random Number Generators. GPU Gems: Emerald Edition., 2011. 231–246

[73] Barnes C, Jacobs D E, Sanders J, et al. Video puppetry: a performative interface for cutout animation. 2008, 27(5):124

[74] Macklin M, Müller M. Position Based Fluids. ACM Transactions on Graphics (TOG), 2013, 32:4

[75] Batty C, Uribe A, Audoly B, et al. Discrete viscous sheets. ACM Transactions on Graphics (TOG), 2012, 31(4):113

[76] Beam R M, Warming R. An implicit factored scheme for the compressible Navier-Stokes equations. AIAA journal, 2012, 16(4)

[77] Zhang F, Wu J, Shen X. SPH-based fluid simulation: a survey. Proceedings of International Conference on Virtual Reality and Visualization. IEEE, 2011. 164–171

[78] Monaghan J. Smoothed particle hydrodynamics and its diverse applications. Annual Review of Fluid Mechanics, 2012, 44:323–346

[79] He X, Liu N, Wang G, et al. Staggered meshless solid-fluid coupling. ACM Transactions on Graphics (TOG), 2012, 31(6):149

[80] Orthmann J, Kolb A. Temporal Blending for Adaptive SPH. 2012, 31(8):2436–2449

[81] Thürey N, Wojtan C, Gross M, et al. A multiscale approach to mesh-based surface tension flows. ACM Transactions on Graphics (TOG), 2010, 29(4):48

[82] Hu X, Adams N. An incompressible multi-phase SPH method. Journal of Computational Physics, 2007, 227(1):264–278

[83] Hu X, Adams N. A constant-density approach for incompressible multi-phase SPH. Journal of Computational Physics, 2009, 228(6):2082–2091

致　谢

首先感谢我的导师，即王斌老师对我的精心指导。在进入大学之前我便选定了图形学作为我的研究方向，然而正统的研究训练还是从大二进入王老师的渲染组开始的，因此我非常感谢王老师接收我入组，在研究最困难的时候给予我深厚的关心，并为我提供了去参加各种实习和SIGGRAPH大会的机会。

另外，感谢香港大学的王文平老师和魏立一老师，以及微软亚洲研究院网络图形组的王律迪、童欣老师；硬件计算组的严婧、赵春水老师给予我实习的机会抑或许多帮助和人生上的指导。我相信，这些指导与经历足以改变我的一生。

同时感谢张慧老师、陈莉老师、刘玉身老师、雍俊海老师给我提供了更多成长的空间，包括任职两年本科图形学课助教以及其他交流机会等，并感谢彭凌老师对我生活上的帮助。

其次需要感谢与我合作过项目的所有老师和同学，包括三星研究院的荣国栋老师、通用电气的刘敏老师、完美世界的姚春晖学长以及网易游戏的王妙一学姐；还有渲染组的所有同学，特别是陈家挺、王宇翔和钱康来等同学，在生活和科研上给了我莫大的帮助。

感谢我的父母及家人，为我提供了一个良好成长的环境，并一直鼎力支持我的学习和生活。

感谢我的朋友和一切爱我的人，为我提供了充溢的精神支持并帮助我成长。

感谢清华大学软件学院这五年来给我的培养和教导。

感谢SIGGRAPH，让我确立了以计算机图形学为信仰。

最后，感谢这个世界，让我目睹了人类的梦的美丽。