Математические структуры и моделирование
2009, вып. 20, с. 52–73

УДК 004.92,532.5,519.633

МОДЕЛИРОВАНИЕ ДИНАМИКИ ФЛЮИДОВ НА ГРАФИЧЕСКОМ ПРОЦЕССОРЕ

Ю.В. Фролова, В.В. Коробицын

An approach to fluid simulation on the GPU is described. The algorithm is based on the physical equations of fluid flow, namely the Navie–Stokes equations. We provide clear explanations and draw connections between the math and its implementation. It describes the techniques to perform the simulation on the GPU.

1. Введение

Флюиды (т.е. жидкости и газы) присутствуют везде: вода в реке, текущая вдоль берега, струйка дыма от сигареты, клубы пара из кипящего чайника, испарения, формирующиеся в облака, смешиваемые краски в банке. В основе всего этого — поток флюидов. Всё это явления, которые многие хотели бы изобразить реалистично в интерактивных графических приложениях. Рисунок 1 показывает примеры смоделированных флюидов.

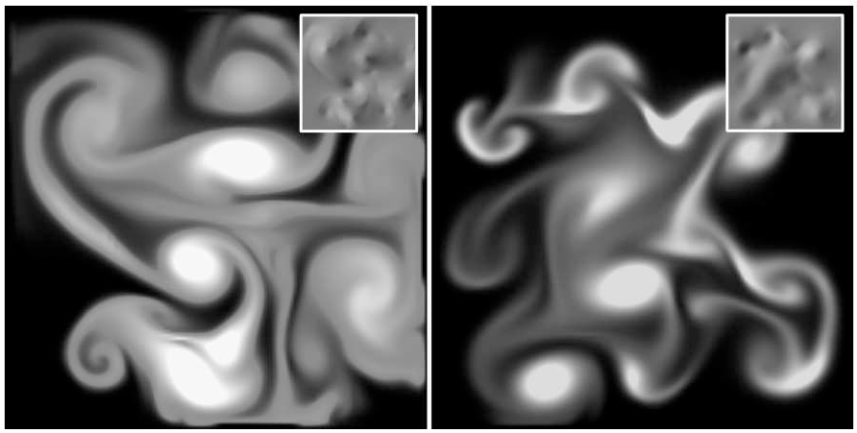

Рис. 1. Имитация вихревого движения краски. В маленькой картинке представлена текстура поля скорости

Copyright © 2009 **Ю.В. Фролова, В.В. Коробицын.**
Омский государственный университет.
E-mail: juliavfrolova@rambler.ru

Моделирование флюидов является полезным кирпичиком, лежащим в основе имитации различных природных явлений. Из-за большого количества параллелизма в графических аппаратных средствах прогоны моделирования значительно быстрее выполняются на графическом процессоре (GPU), чем на центральном процессоре (CPU). Используя современные видеокарты NVIDIA GeForce, можно достигнуть ускорения в несколько раз над эквивалентным моделированием на CPU.

Цель представленной работы состоит в раскрытии возможностей и помощи в изучении мощного средства. Без понимания основ физики и математики флюидов использование и развитие алгоритмов не представляется возможным. По этой причине в работе подробно рассмотрена математическая часть, содержащая много уравнений. По возможности даются ясные объяснения и показываются связи между математическими формулами и компьютерной реализацией. Для написания данной статьи были использованы работы М. Харриса [5–8], Дж. Крюгера и Р. Вестерманна [9], В. Ли, Ж. Фана, К. Вейя и А. Кауфмана [10], Дж. Стама [11, 12].

Предполагается, что читатель знаком с курсами дифференциального и интегрального исчисления, дифференциальных уравнений и векторного исчисления. Будет полезен опыт с конечно-разностным приближением производных.

Описываемая методика основана на методе «устойчивых флюидов» [11]. Однако Дж. Стам моделировал, используя реализацию на CPU, здесь же будет описана реализация на графических аппаратных средствах. GPU хорошо подходит для вычислений, требующихся для моделирования флюидов. Представляемая модель выполнена на *сетке ячеек*. Программируемые GPU оптимизированы для выполнения вычислений над пикселями, которые можно рассматривать как ячейки сетки. GPU достигает высокой скорости вычислений за счет параллелизма: способность обрабатывать огромное число вершин и пикселей одновременно. Также оптимизировано выполнение многократного поиска по текстуре за один цикл. Поскольку имитационная сетка будет помещаться на текстурах, то скорость и параллелизм — это то, что нам нужно.

Эта статья не раскрывает все вопросы гидродинамики. Ограничимся моделированием только непрерывного объема флюида в двумерной прямоугольной области. Также не имитируются свободные границы поверхности между флюидами, как, например, граница раздела между всплесками воды и воздуха.

Во всей работе используются единые математические обозначения. В уравнениях *наклонный* шрифт используется для обозначения переменных, которые представляют скалярные величины, например давление p. **Жирный** шрифт используется для представления векторных величин, например скорость $\mathbf{u}$. Все векторы в этой статье двумерны.

В пункте 2 описывается математическая основа метода, включая рассмотрение уравнений, которые управляют потоком флюида. Затем обсуждается подход к решению уравнений. Пункт 3 описывает реализацию моделирования флюида на GPU. Пункт 4 представляет некоторые примеры моделирования флюидов.

2. Математическая основа

Для имитации поведения флюида необходимо иметь математическое описание состояния флюида в каждый момент времени. Наиболее важным параметром для представления флюида является его скорость, поскольку скорость определяет как перемещение самого флюида, так и предметов, которые в нем находятся. Скорость флюида изменяется как во времени, так и в пространстве, поэтому представляется как векторное поле.

Векторное поле есть некоторое отображение векторнозначной функции в параметризованное пространство, например декартову сетку. (Другие параметризации пространства также возможны, но мы принимаем двумерную декартову сетку.) Векторное поле скорости флюида определено из условия, что для каждой позиции $\mathbf{x} = (x, y)$, есть сопоставленная скорость, изменяющаяся во времени t, $\mathbf{u}(\mathbf{x}, t) = (u(\mathbf{x}, t), v(\mathbf{x}, t))$, как показано на рисунке 2.

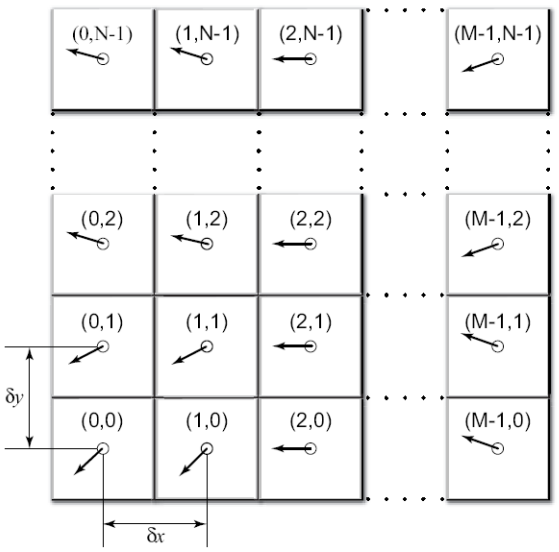

Рис. 2. Векторная сетка флюида

Ключевым для моделирования флюида являются шаги во времени, на каждом временном шаге нужно правильно определить текущее поле скорости. Это можно сделать решая уравнения, которые описывают изменения поля скоростей со временем под действием ряда сил. Как только у нас будет поле скоростей, мы можем сделать интересные вещи с ним, например, используем его для перемещения объектов или для отображения плотности дыма.

2.1. Уравнения движения Навье-Стокса

Флюид является несжимаемым, если масса любой его части постоянна во времени. Флюид однороден, если его плотность ρ не изменяется в пространстве. Комбинация несжимаемости и однородности означает, что плотность является константой как с течением времени, так и в пространстве. Эти предположения

являются общими в гидродинамике и применяются для моделирования реальных флюидов, например воды и воздуха.

Рассматриваемая модель флюида реализуется на регулярной декартовой сетке с пространственными координатами $\mathbf{x} = (x, y)$ и переменной времени t. Флюид представлен полем скоростей $\mathbf{u}(\mathbf{x}, t)$ и скалярным полем давлений $p(\mathbf{x}, t)$. Эти поля изменяются как в пространстве, так и во времени. Если скорость и давление известны в начальный момент времени $t = 0$, тогда состояние флюида во времени может быть описано уравнениями Навье-Стокса для несжимаемого потока:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{F}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2}$$

где ρ — (константа) плотность флюида, ν — кинематическая вязкость и вектор $\mathbf{F} = (f_x, f_y)$ представляет любые внешние силы, которые действуют на флюид. Заметим, что уравнение (1) — это два уравнения, поскольку $\mathbf{u}$ — векторная величина:

$$\frac{\partial u}{\partial t} = -(\mathbf{u} \cdot \nabla)u - \frac{1}{\rho}\nabla p + \nu \nabla^2 u + f_x,$$

$$\frac{\partial v}{\partial t} = -(\mathbf{u} \cdot \nabla)v - \frac{1}{\rho}\nabla p + \nu \nabla^2 v + f_y.$$

Таким образом, имеем три неизвестных u, v и p и три уравнения.

Уравнения Навье-Стокса легче понять, разбирая их на простые части. Сначала попытаемся понять члены, влияющие на поток флюида, их четыре в правой части уравнения (1). Рассмотрим каждый в отдельности.

2.2. Члены в уравнениях Навье-Стокса

Перемещение потока. Скорость флюида способствует перемещению объектов, плотностей и других величин вместе с потоком. Представим попадание струи краски в движущуюся жидкость. Краска перемещается вдоль поля скорости флюида. Фактически скорость флюида несет флюид параллельно, подобно тому как она несет краску. Первый член из правой части уравнения (1) представляет собой само-адвекцию поля скоростей и назван *адвекцией* (перемещением потока).

Давление. Поскольку молекулы флюида могут перемещаться вокруг друг друга, они стремятся к «всплескам» и «хлюпам». Когда некоторая сила приложена к флюиду, она немедленно распространяется через весь объем. Молекулы, близкие к месту приложения силы, наталкиваются на те, которые дальше от него, и создается давление. Поскольку давление есть сила на единицу области, любое давление во флюиде естественно ведет к ускорению. (Вспомним второй закон Ньютона, $F = ma$.) Второй член уравнения, названный *давлением*, представляет ускорение флюида.

Диффузия. Из опыта с реальными флюидами известно, что некоторые жидкости «плотнее» других. Например, сахарный сироп струится медленнее, а этиловый спирт быстрее воды. Говорят, что более плотные флюиды имеют более высокую *вязкость*. Вязкость является мерой того, как жидкость сопротивляется течению. Это сопротивление заканчивается диффузией движущей силы (следовательно, и скорости), так что третий член называем *диффузией*.

Внешние силы. Четвертый член представляет ускорение, возникающее в результате действия внешних сил, приложенных к флюиду. Эти силы могут быть либо *локальными*, либо *объектными* силами. Локальные силы приложены к отдельной области флюида, например сила вентилятора, создающего поток воздуха. Объектная сила прикладывается ко всему флюиду, например сила тяжести.

2.3. Краткий обзор векторного исчисления

Уравнения (1) и (2) содержат символ ∇, известный как набла-оператор. Три способа применения набла-оператора — градиент, дивергенция и оператор Лапласа – приведены в таблице 1. В столбце «символ» приводятся обозначения операторов, принятые в отечественной литературе. Индексы i и j, использованные в выражениях в таблице, указывают на дискретные позиции на декартовой сетке, δx и δy — шаги сетки по осям x и y соответственно.

Таблица 1. Операторы векторного исчисления, используемые в гидродинамике

оператор	определение	символ	конечно-разностная форма
Градиент	$\nabla p = \left(\frac{\partial p}{\partial x}, \frac{\partial p}{\partial y}\right)$	grad p	$\frac{p_{i+1,j}-p_{i-1,j}}{2\delta x}, \frac{p_{i,j+1}-p_{i,j-1}}{2\delta y}$
Дивергенция	$\nabla \cdot \mathbf{u} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$	div $\mathbf{u}$	$\frac{u_{i+1,j}-u_{i-1,j}}{2\delta x} + \frac{v_{i,j+1}-v_{i,j-1}}{2\delta y}$
Лапласа	$\nabla^2 p = \frac{\partial^2 p}{\partial x^2} + \frac{\partial^2 p}{\partial y^2}$	$\triangle p$	$\frac{p_{i+1,j}-2p_{i,j}+p_{i-1,j}}{(\delta x)^2} + \frac{p_{i,j+1}-2p_{i,j}+p_{i,j-1}}{(\delta y)^2}$

Градиент скалярного поля есть вектор частных производных скалярного поля. Дивергенция, которая появляется в уравнении (2), имеет важное физическое значение. Это интенсивность источника потока из данной точки пространства. В уравнениях Навье-Стокса оператор дивергенции приложен к скорости потока и показывает изменение скорости через поверхность, окружающую небольшую часть флюида. Уравнение неразрывности (2) предполагает несжимаемость флюида, то есть флюид должен иметь нулевую дивергенцию. Скалярное произведение в операторе дивергенции дает в результате сумму частных производных, а не вектор, как оператор градиента. Это означает, что оператор дивергенции может прилагаться только к векторному полю, например скорости $\mathbf{u} = (u, v)$.

Отметим, что градиент скалярного поля — векторное поле, а дивергенция векторного поля является скалярным полем. Если оператор дивергенции при-

ложен к результату оператора градиента, то результатом является оператор Лапласа $\nabla \cdot \nabla = \nabla^2$. Для упрощения примем, что ячейки сетки квадратные ($\delta x = \delta y$), тогда оператор Лапласа имеет конечно-разностную форму

$$\nabla^2 p = \frac{p_{i+1,j} + p_{i-1,j} + p_{i,j+1} + p_{i,j-1} - 4p_{i,j}}{(\delta x)^2}. \tag{3}$$

Оператор Лапласа встречается в физике в форме уравнений диффузии, например уравнение теплопроводности. Уравнение вида $\nabla^2 x = b$ известно как уравнение Пуассона. Когда $b = 0$ — уравнение Лапласа. В уравнении (1) оператор Лапласа приложен к векторному полю. Это символьное упрощение: оператор прилагается отдельно к каждому скалярному компоненту векторного поля.

2.4. Решение уравнения Навье-Стокса

Уравнения Навье-Стокса можно решить аналитически только для некоторых простых физических форм. Тем не менее, можно использовать методы численного интегрирования для пошагового решения. Поскольку нас интересует наблюдение динамики потока во времени, то пошаговое численное решение нам вполне подходит.

Как с любым алгоритмом, решение уравнений Навье-Стокса необходимо разбить на простые шаги. Используемый нами метод основан на методике устойчивых флюидов, описанной в [11]. Здесь приводится описание каждого шага, а в пункте 3 описывается их реализация с использованием языка Cg для GPU.

Сначала необходимо преобразовать уравнение к виду, приемлемому для численного решения. Вспомним, что уравнения Навье-Стокса — это три скалярных уравнения, которые необходимо решить для поиска величин u, v и p. Тем не менее, поиск решения не очевиден. Далее описывается преобразование, ведущее к простому алгоритму.

Разложение Гельмгольца-Ходжа. Любой вектор $\mathbf{v}$ может быть разложен по базисным векторам. Например, векторы на декартовой сетке представляются как пара отрезков вдоль осей сетки: $\mathbf{v} = (x, y)$. Тот же вектор может быть записан: $\mathbf{v} = x\mathbf{i} + y\mathbf{j}$, где $\mathbf{i}$ и $\mathbf{j}$ — базисные вектора, направленные вдоль осей сетки.

Так же можно разложить вектор на сумму векторов, а поле векторов — на сумму векторных полей. Пусть D — область в пространстве или в нашем случае на плоскости, в которой определен наш флюид. Пусть данная область имеет гладкую границу ∂D (то есть дифференцируемую) с вектором нормали n. Можно использовать теорему Гельмгольца–Ходжа о разложении вектора, приведенную в [2].

Теорема о разложении Гельмгольца-Ходжа. *Векторное поле* $\mathbf{w}$ *на* D *можно однозначно разложить на составные части вида:*

$$\mathbf{w} = \mathbf{u} + \nabla p, \tag{4}$$

где $\mathbf{u}$ *имеет нулевую дивергенцию и параллельно к* ∂D*, то есть* $\mathbf{u} \cdot \mathbf{n} = 0$ *на* ∂D*.*

Теорема приводится без доказательства. Подробнее смотрите [2, с. 37–38].

Эта теорема указывает, что любое векторное поле может быть разложено на сумму двух других векторных полей: векторное поле с нулевой дивергенцией и градиент скалярного поля. При этом поле с нулевой дивергенцией стремится к нулю на границе. Это мощное средство, ведущее нас к двум полезным реализациям.

Первая реализация. Численное решение уравнений Навье-Стокса разделяется на три этапа вычисления скорости на каждом временном шаге: адвекция, диффузия и приложение сил. Результатом является новое поле скорости $\mathbf{w}$ с ненулевой дивергенцией. Но уравнение неразрывности требует, чтобы мы завершали временной шаг со скоростью с нулевой дивергенцией. К счастью, теорема о декомпозиции Гельмгольца-Ходжа утверждает, что дивергенция скорости может быть скорректирована вычитанием градиента получившегося поля давления

$$\mathbf{u} = \mathbf{w} - \nabla p. \tag{5}$$

Вторая реализация. Теорема также дает метод для вычисления поля давления. Если применить оператор дивергенции к обеим частям уравнения (4), то получим

$$\nabla \cdot \mathbf{w} = \nabla \cdot (\mathbf{u} + \nabla p) = \nabla \cdot \mathbf{u} + \nabla^2 p. \tag{6}$$

В силу уравнения (2), $\nabla \cdot \mathbf{u} = 0$, упростим

$$\nabla^2 p = \nabla \cdot \mathbf{w}, \tag{7}$$

а это есть уравнение Пуассона (см. п. 2.3) для давления флюида. Это означает, что после перехода к дивергенции скорости $\mathbf{w}$ можно решить уравнение (7) для p, затем использовать $\mathbf{w}$ и p для вычисления нового поля $\mathbf{u}$ с нулевой дивергенцией, используя уравнение (5). Обратимся к этому позже.

Теперь нам необходим способ для вычисления $\mathbf{w}$. Вернемся к сопоставлению векторов и векторных полей. Из определения скалярного произведения известно, что можно найти проекцию вектора $\mathbf{r}$ на единичный вектор $\mathbf{s}$, вычислив скалярное произведение $\mathbf{r}$ и $\mathbf{s}$. Скалярное произведение есть оператор проекции для векторов, который отображает вектор $\mathbf{r}$ на свою компоненту в направлении $\mathbf{s}$. Можно использовать теорему о декомпозиции Гельмгольца-Ходжа для определения оператора проекции $\mathbb{P}$, который проектирует векторное поле $\mathbf{w}$ на векторное поле $\mathbf{u}$ с нулевой дивергенцией. Если применить оператор $\mathbb{P}$ к уравнению (4), получим

$$\mathbb{P}\mathbf{w} = \mathbb{P}\mathbf{u} + \mathbb{P}(\nabla p).$$

Но по определению $\mathbb{P}$, $\mathbb{P}\mathbf{w} = \mathbb{P}\mathbf{u} = \mathbf{u}$. Следовательно, $\mathbb{P}(\nabla p) = 0$. Теперь используем эти идеи для упрощения уравнения Навье-Стокса.

Сначала применим оператор проекции к обеим частям уравнения (1):

$$\mathbb{P}\frac{\partial \mathbf{u}}{\partial t} = \mathbb{P}\left(-(\mathbf{u}\cdot\nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{F}\right).$$

Поскольку $\mathbf{u}$ имеет нулевую дивергенцию, то производная в левой части уравнения $\mathbb{P}(\partial\mathbf{u}/\partial t) = \partial\mathbf{u}/\partial t$, а также $\mathbb{P}(\nabla p) = 0$, таким образом, член давления исчезает. Остается уравнение в следующем виде

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbb{P}\left(-(\mathbf{u}\cdot\nabla)\mathbf{u} + \nu\nabla^2\mathbf{u} + \mathbf{F}\right). \tag{8}$$

Важно, что это уравнение заключает в себе последовательность шагов алгоритма моделирования потока флюида. Сначала вычислим то, что находится в круглых скобках в правой части уравнения. Последовательно вычисляем адвекцию, диффузию и приложение сил. Это дает в результате дивергенцию поля скоростей $\mathbf{w}$, к которому применяем оператор проекции, чтобы получить новое поле $\mathbf{u}$ с нулевой дивергенцией. Для этого решаем уравнение (7) для p, затем вычитаем градиент p из $\mathbf{w}$, как в выражении (5).

Обычно при реализации, компоненты не вычисляются отдельно и не складываются, как в уравнении (8). Взамен этого решение находится как последовательность преобразований векторного поля. Другими словами, каждый компонент есть шаг, который берет поле на вход и производит новое поле на выход. Можно определить оператор $\mathbb{S}$, который эквивалентен решению уравнения (8) за один шаг по времени. Оператор определяется как последовательность операторов адвекции ($\mathbb{A}$), диффузии ($\mathbb{D}$), приложение сил ($\mathbb{F}$) и проекция ($\mathbb{P}$):

$$\mathbb{S} = \mathbb{P}\circ\mathbb{F}\circ\mathbb{D}\circ\mathbb{A}. \tag{9}$$

Таким образом, шаг алгоритма моделирования может быть выражен

$$\mathbb{S}(\mathbf{u}) = \mathbb{P}\circ\mathbb{F}\circ\mathbb{D}\circ\mathbb{A}(\mathbf{u}).$$

Операторы применяются справа налево; первый – адвекция, за которым следует диффузия, приложение сил и проекция. Заметим, что время опускается здесь для ясности, но на практике шаг по времени должен быть использован в вычислении каждого оператора. Теперь разберем более внимательно шаги адвекции и диффузии, а затем подойдем к решению уравнений Пуассона.

Перемещение потока. Адвекция — процесс, посредством которого скорость флюида перемещает флюид и другие величины во флюиде. Чтобы вычислить результат адвекции, необходимо скорректировать значения величины в каждой точке сетки. Поскольку мы вычисляем, как величина перемещается вдоль поля скорости, то легче представить каждую ячейку сетки как частицу. Сначала попытаемся вычислить результат адвекции, для этого обновим значения величин на сетке так, как будто двигается система частиц. Переместим позицию $\mathbf{r}$ каждой частицы вперед вдоль поля скоростей на расстояние, которое она пройдет за время dt:

$$\mathbf{r}(t+\delta t) = \mathbf{r}(t) + \mathbf{u}(t)\delta t.$$

Вы можете понимать это как метод Эйлера. Это простой метод для явного интегрирования обыкновенных дифференциальных уравнений. Имеются более точные методы, например метод средней точки или методы Рунге-Кутты.

Имеется две проблемы с этим подходом: первая это то, что явные методы неустойчивы при больших шагах по времени, и решение может уйти в бесконечность («взорваться»), если величина $\mathbf{u}(t)dt$ больше размера одной ячейки сетки. Вторая проблема характерна для реализаций на GPU. При моделировании с использованием фрагментных шейдеров невозможно изменить позиции фрагментов. А этот метод интегрирования требует возможности «перемещать» частицы, что не может быть реализовано на имеющихся GPU.

Для решения первой проблемы воспользуемся неявным методом [11]. Прежде чем вычислить адвекцию величины, переместим частицу из текущей позиции в предыдущую, используя текущую скорость и отрицательный шаг по времени. Вычисляем величину в этой точке и переносим ее в стартовую ячейку сетки. Чтобы обновить величину q (это может быть скорость, плотность, температура или любая величина, несомая флюидом), используем следующее уравнение:

$$q\Big(\mathbf{x}, t+\delta t\Big) = q\Big(\mathbf{x} - \mathbf{u}(\mathbf{x}, t)\delta t, t\Big). \tag{10}$$

Не только легко реализовать этот метод на GPU, но и, как показал Дж. Стам, решение будет устойчивым для произвольных шагов по времени и любых скоростей. На рисунке 3 представлено вычисление адвекции в ячейке, выделенной двойным кругом. Поле скорости перемещается назад во времени на вектор $\mathbf{x}$. С помощью билинейной интерполяции четырех величин сетки, соседних к $\mathbf{x}$ (помечены квадратом), получаем результат, который записывается в стартовую ячейку сетки.

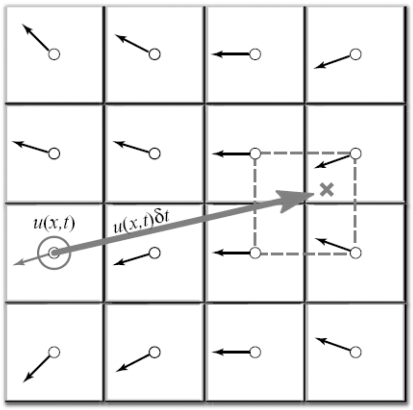

Рис. 3. Вычисление адвекции флюида

Диффузия вязкости. Как отмечалось ранее, вязкие флюиды сопротивляются перемещению в потоке, что является результатом диффузии (или рассеивания) скорости. Дифференциальное уравнение в частных производных для

диффузии вязкости

$$\frac{\partial \mathbf{u}}{\partial t} = \nu \nabla^2 \mathbf{u}. \tag{11}$$

Так же, как для адвекции, есть несколько методов решения этого уравнения. Простой подход состоит в использовании явной схемы

$$\mathbf{u}(\mathbf{x}, t + \delta t) = \mathbf{u}(\mathbf{x}, t) + \nu \delta t \nabla^2 \mathbf{u}(\mathbf{x}, t).$$

В этом уравнении ∇^2 — дискретная форма оператора Лапласа (3). Подобно явному методу Эйлера для вычисления адвекции, этот метод неустойчив для больших величин δt и ν [12]. Поэтому, следуя работе Дж. Стама, снова используем неявную схему для уравнения (11):

$$\left(\mathbf{I} - \nu \delta t \nabla^2\right) \mathbf{u}(\mathbf{x}, t + \delta t) = \mathbf{u}(\mathbf{x}, t), \tag{12}$$

где $\mathbf{I}$ — единичная матрица. Эта схема устойчива для произвольного шага по времени и любой вязкости. Это уравнение есть уравнение Пуассона для скорости. Напомним, что использование декомпозиции Гельмгольца-Ходжа заканчивается уравнением Пуассона для давления. Оба этих уравнения могут быть решены итерационным методом релаксации.

Решение уравнения Пуассона. Необходимо решить два уравнения Пуассона: для давления и для диффузии вязкости. Уравнения Пуассона хорошо изучены. Используем итерационный метод решения, начиная с начального приближения, улучшаем решение на каждой итерации.

Дискретный аналог уравнения Пуассона является системой линейных алгебраических уравнений $\mathbf{Ax} = \mathbf{b}$, где $\mathbf{x}$ – вектор величин, для которых ищем решение (p или $\mathbf{u}$ в нашем случае), $\mathbf{b}$ – вектор констант, $\mathbf{A}$ – матрица. В нашем случае $\mathbf{A}$ представляет оператор Лапласа ∇^2, так что нет необходимости хранить $\mathbf{A}$ как матрицу. Итерационный метод решения стартует с начального приближения решения $\mathbf{x}^{(0)}$, и на каждом шаге k вычисляется новое улучшенное решение $\mathbf{x}^{(k)}$. Надстрочным индексом указывается номер итерации. Пожалуй, самым простым итерационным методом является итерационный метод Якоби. Вид итерации Якоби для СЛАУ можно найти в [3].

Более изощренные методы, такие как метод сопряженных градиентов и многосеточный метод, сходятся быстрее, но здесь используется метод Якоби из-за простоты реализации. Другие методы можно найти в литературе [1, 4, 9].

Уравнения (7) и (12) выглядят по-разному, но оба могут быть дискретизированы, используя выражение (3), и записаны в виде

$$x_{i,j}^{(k+1)} = \frac{x_{i-1,j}^{(k)} + x_{i+1,j}^{(k)} + x_{i,j-1}^{(k)} + x_{i,j+1}^{(k)} + \alpha b_{i,j}}{\beta}, \tag{13}$$

где α и β – константы. Значения x, b, α и β различны для двух уравнений. В уравнении Пуассона для давления x представляет p, b представляет $\nabla \cdot \mathbf{w}$, $\alpha =$

$-(\delta x)^2$ и $\beta = 4$. Для уравнения диффузии вязкости как x, так и b представляют $\mathbf{u}$, $\alpha = (\delta x)^2/(\nu\delta t)$ и $\beta = 4 + \alpha$.

Приведение двух уравнений к одному виду позволяет использовать один исходный код для решения обоих уравнений. Чтобы решить уравнения, просто запускаем множество итераций, в которых применяется формула (13) к каждой ячейке сетки. При этом используются результаты предыдущей итерации $\mathbf{x}^{(k)}$ для вычисления следующей $\mathbf{x}^{(k+1)}$. Так как итерационный метод Якоби сходится медленно, то необходимо выполнить много итераций. Поскольку итерации метода Якоби просты в исполнении на GPU, то можно выполнить много итераций в течение короткого момента времени.

Начальные и граничные условия. Любая задача с дифференциальными уравнениями, определенная в конечной области, требует задания граничных условий. Граничные условия определяются исходя из предположения о способе изменения значений на границе области моделирования. Кроме того, для решения задачи, меняющейся со временем, необходимо задать начальные условия. В нашем случае в начальный момент времени флюид имеет нулевую скорость и нулевое давление в каждой точке. Граничные условия требуют дальнейшего обсуждения.

На каждом шаге решаются уравнения для двух величин — скорости и давления, поэтому необходимы граничные условия для обоих уравнений. Поскольку флюид моделируется на прямоугольной сетке, то предполагается, что флюид находится внутри ящика и не может протечь сквозь стороны ящика. Для скорости на границе используем условие непротекания флюида сквозь границы, что задается стремлением скорости к нулю на границах. Корректное решение уравнения Пуассона для давления требует граничного условия Неймана: $\partial p/\partial n = 0$. Это означает, что на границе, скорость изменения давления в направлении нормали к границе равна нулю. Снова вернемся к граничным условиям в конце пункта 3.

3. Реализация

Теперь, когда понятен способ решения задачи, можно перейти к реализации. Приведем псевдокод алгоритма вычисления одного шага по времени. Переменные u и p предназначены для хранения поля скорости и поля давления соответственно.

```
// применим последовательно 3 оператора из уравнения (9).
 u = advect(u);
 u = diffuse(u);
 u = addForces(u);
// затем применим к результату оператор проекции.
 p = computePressure(u);
 u = subtractPressureGradient(u, p);
```

На практике необходимо использовать дополнительную переменную для выполнения операций. Например, псевдокод шага вычисления адвекции:

```
uTemp = advect(u);
swap(u, uTemp);
```

Этот псевдокод не содержит специфических деталей реализации. Фактически этот псевдокод применим для реализации и на CPU, и на GPU. Наша основная цель выполнить все шаги на GPU. Вычисление такого рода на GPU имеют свою специфику, поэтому проведем некоторые аналогии между действиями при моделировании флюидов на CPU и на GPU.

3.1. CPU–GPU-аналогии

Основой любого компьютера являются модели памяти и процессора, следовательно, любое приложение должно учитывать способы представления данных и вычислений. Рассмотрим различия между CPU и GPU в этом отношении.

Текстуры = Массивы. Рассматриваемая модель флюида представляет данные на двумерной сетке. Естественное представление этой сетки на CPU — массив. Аналог массива на GPU — текстура. Хотя текстуры не такие гибкие, как массивы, но их гибкость постоянно улучшается с развитием графических аппаратных средств. Текстуры на современных GPU поддерживают все основные операции, необходимые для осуществления моделирования флюида. Поскольку текстуры обычно имеют три или четыре цветовых канала, то они обеспечивают естественное представление векторных типов данных от двух до четырех компонент. Кроме того, многочисленные скалярные поля могут быть сохранены в одной текстуре. Основная операция — чтение из массива (или памяти), которая выполняется с использованием операции просмотра текстуры. Таким образом, аналогом индекса массива является координата в текстуре на GPU. Необходимы, по крайней мере, две текстуры для представления состояния флюида: одна для скорости и одна для давления. Для отображения потока необходима дополнительная текстура, которая содержит величины, перемещаемые флюидом.

Тело цикла = Фрагмент программы. При реализации на CPU в цикле выполняется множество шагов алгоритма, используя пару вложенных циклов для повторения операций над каждой ячейкой в сетке. В каждой ячейке сетки выполняются одинаковые вычисления. GPU не имеет возможности выполнять внутренний цикл над каждым текселем (элемент текстуры) в текстуре. Тем не менее, фрагментный конвейер предназначен для выполнения одинаковых вычислений на каждом фрагменте. Программист представляет это, как будто есть отдельный процессор для каждого фрагмента, и все фрагменты обновляются одновременно. На языке параллельного программирования эта модель известна как SIMD (один поток команд - много потоков данных). Таким образом, аналог обработки вложенных циклов над массивом — фрагментная программа на GPU, применяемая к каждому фрагменту методом SIMD.

Обратная связь = Коррекция текстуры. В пункте 2.4 описано, как используется итерация Якоби для решения уравнения Пуассона. Этот итерационный метод использует результат итерации как входные данные для следующей итерации. Такая обратная связь обычна для численных методов. При реализации на CPU обратная связь тривиально осуществляется с использованием переменных и массивов, к которым имеется доступ на чтение и запись. Фрагментный процессор на GPU всегда записывает результат в буфер кадров. Кадровый буфер можно представить себе как двумерный массив, который не может быть напрямую прочитан. Есть два пути получить содержание кадрового буфера в текстуру, которая может быть прочитана:

- *скопировать в текстуру* (CTT) — содержание кадрового буфера копируется в текстуру;
- *рендер в текстуру* (RTT) — используется текстура как кадровый буфер, куда GPU может напрямую записывать.

Функции CTT и RTT одинаково хороши. Выбор функции зависит от конкретной реализации. Далее считается, что реализована операция записи в текстуру.

Ранее упоминалось, что на практике каждые из пяти шагов алгоритма заносят результат во временную сетку, затем выполняется перестановка сеток (свопинг). Функция RTT требует использования двух текстур для осуществления обратной связи, поскольку результаты чтения из текстуры не определены, пока происходит рендеринг. Перестановка в этом случае является просто обменом идентификаторов текстур. Следовательно, временные затраты функции RTT постоянны. С другой стороны, функция CTT требует только одной текстуры. Кадровый буфер используется как временная сетка, и перестановка может быть выполнена копированием данных из кадрового буфера в текстуру. Временные затраты этого копирования пропорциональны размеру текстуры.

3.2. Элементарные операции

Шаги моделирования поделим на элементарные операции. Каждая такая операция включает в себя обработку одного или более фрагментов (зачастую всех) в кадровом буфере активной фрагментной программой, завершаемой обновлением текстуры. Обработка фрагмента управляется рендерингом геометрических примитивов. Используемые в этом приложении примитивы просты: либо прямоугольник, либо отрезок.

Существует два типа фрагментов для обработки в каждой элементарной операции: внутренние фрагменты и граничные фрагменты. Наша двумерная сетка имеет границу, составленную из периметра, шириной в одну ячейку. Обычно вычисления различны для ячеек внутри и на границе. Для обновления внутренних фрагментов рисуется прямоугольник, покрывающий весь кадровый буфер, за исключением границы, шириной в одну ячейку. Граница выводится с помощью четырех отрезков. Реализуются отдельные фрагментные программы для внутренних и граничных ячеек. Смотри рисунок 4.

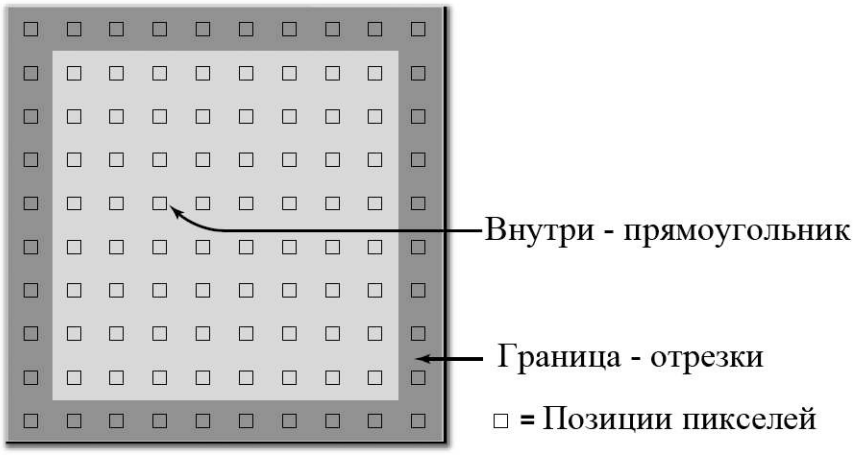

Рис. 4. Примитивы, используемые для обновления внутренних и граничных ячеек сетки

3.3. Реализация фрагментных программ

Теперь мы знаем шаги алгоритма, представление данных и как выполняются элементарные операции. Приступим к написанию фрагментных программ для выполнения вычислений в каждой ячейке.

Перемещение потока. Реализация фрагментной программы для адвекции приведена в листинге 1, которая почти повторяет уравнение (10). Имеется одно незначительное отличие. Поскольку координаты текстуры изменяются в диапазоне отличном от координат области моделирования (координаты текстуры — в диапазоне $[0, N]$, где N — разрешение сетки), необходимо масштабировать скорость в пространство сетки. Это отражено в коде Cg с помощью умножения локальной скорости на параметр rdx, который представляет обратную величину шага сетки δx. Способ заворачивания текстуры должен быть установлен на CLAMP_TO_EDGE, чтобы значения, выходящие за пределы диапазона $[0, N]$, были прикреплены к граничным текселям. Граничные условия корректно обрабатывают эти тексели, как будет описано ниже.

Листинг 1. *Фрагментная программа для адвекции*

```
void advect(float2 coords   : WPOS,   // координаты сетки
   out float4 xNew : COLOR,  // результат адвекции qty
   uniform float timestep,
   uniform float rdx,        // 1 / dx
   uniform samplerRECT u,    // входное поле скорости
   uniform samplerRECT x)    // qty для адвекции
{
  // учитывая поле скорости, делаем "шаг назад"
   float2 pos = coords - timestep * rdx * f2texRECT(u, coords);

  // интерполируем и записываем результат
  xNew = f4texRECTbilerp(x, pos);
}
```

В этом коде параметр u является текстурой поля скорости, а x — поле, которое будет получено после адвекции. Это может быть скоростью или любой другой величиной, например концентрация краски. Функция f4texRECTbilerp() выполняет билинейную интерполяцию четырех текселей, ближайших к точке с вычисленными текстурными координатами. Поскольку современные GPU не поддерживают автоматическую билинейную интерполяцию в текстурах со значениями с плавающей точкой, то необходимо осуществить это с помощью данной функции.

Диффузия вязкости. Описание итерации метода Якоби, приведенное выше в пункте 2.4, реализовано фрагментной программой в листинге 2.

Листинг 2. *Фрагментная программа итерации Якоби для решения уравнения Пуассона*

```
void jacobi(half2 coords   : WPOS,   // координаты на сетке
   out
   half4 xNew : COLOR,  // результат
   uniform half alpha,
   uniform half rBeta,      // 1/ beta
   uniform samplerRECT x,   // вектор x (Ax = b)
   uniform samplerRECT b)   // вектор b (Ax = b)
{
  // левая, правая, нижняя и верхняя ячейки к x
  half4 xL = h4texRECT(x, coords - half2(1, 0));
  half4 xR = h4texRECT(x, coords + half2(1, 0));
  half4 xB = h4texRECT(x, coords - half2(0, 1));
  half4 xT = h4texRECT(x, coords + half2(0, 1));

  // ячейка b из центра
   half4 bC = h4texRECT(b, coords);

  // выполнение итерации Якоби
  xNew = (xL + xR + xB + xT + alpha * bC) * rBeta;
}
```

Заметим, что параметр rBeta — это обратная величина к β из формулы (13). Для решения уравнения диффузии устанавливается alpha$= (\delta x)^2/(\nu\delta t)$, rBeta$= 1/\big(4 + (\delta x)^2/(\nu\delta t)\big)$, а также параметры x и b на текстуру поля скорости. Затем выполняется некоторое количество итераций (обычно от 20 до 50, большее количество можно использовать, чтобы уменьшить ошибку).

Приложение сил. Самый простой шаг алгоритма – это вычисление ускорения под действием внешних сил. В моделирующем приложении импульс внешней силы можно прикладывать к флюиду посредством нажатия и перемещения мыши. Для реализации этого рисуем пятно в текстуре поля скоростей в позиции

щелчка мыши. Цвет пятна кодирует силу и направление импульса: красный канал содержит величину вдоль оси x, а зеленый канал содержит величину вдоль оси y. Пятно представляется двумерной Гауссовской «шапочкой».

Используем фрагментную программу для вычисления расстояния до точки приложения импульса. Затем добавляем величину $\mathbf{c}$ к цвету

$$\mathbf{c} = \mathbf{F}\delta t \exp\left(\frac{(x - x_p)^2 + (y - y_p)^2}{r}\right),$$

здесь $\mathbf{F}$ — сила, вычисленная в зависимости от расстояния и направления перемещения мыши, r — заданный радиус импульса, (x, y) и (x_p, y_p) — позиции фрагмента и импульса (щелчка мыши) в координатах окна соответственно.

Проекция. В начале этого пункта указывалось, что оператор проекции реализуется за две операции: решение уравнения Пуассона для поля давления p и вычитание градиента p из промежуточного поля скорости. Для этого потребуются три фрагментные программы: вышеупомянутая программа итерации Якоби, программа для вычисления дивергенции промежуточного поля скорости, программа для вычитания градиента p из промежуточного поля скорости.

Программа для вычисления дивергенции приведена в листинге 3. В качестве входных параметров для нее требуются промежуточное поле скорости w и половина обратной величины шага сетки halfrdx. Она вычисляет дивергенцию согласно конечно-разностной формуле, приведенной в таблице 1.

Результат дивергенции записывается во временную текстуру, которая затем используется как входной параметр b в программе итерации Якоби. Параметр x в программе итерации Якоби устанавливается в текстуру поля давления, которая первоначально имеет все нулевые значения (другими словами, нуль используется как начальное предположение для поля давления). Параметры alpha и rBeta установлены как $-(\delta x)^2$ и $1/4$ соответственно.

Для достижения хорошей сходимости решения будем использовать от 40 до 80 итераций Якоби. Изменение количества итераций Якоби будет влиять на точность моделирования. Если количество итераций меньше 20, то ошибка уже заметна. Использование большего количества итераций приводит к более подробной детализации и повышению точности, но требует больших временных затрат для вычисления. После завершения итерации Якоби текстура поля давления передается как параметр p в следующую программу, которая вычисляет градиент p согласно определению из таблицы 1 и вычитает его из промежуточного поля скорости, хранящегося в текстуре w. Смотри листинг 4.

Листинг 3. *Фрагментная программа вычисления дивергенции*

```
void divergence(half2 coords  : WPOS,   // координаты на сетке
   out
   half4 div : COLOR,  // дивергенция
   uniform half halfrdx,   // 0.5 / шаг сетки
   uniform samplerRECT w)  // векторное поле
{
```

```
  half4 wL = h4texRECT(w, coords - half2(1, 0));
  half4 wR = h4texRECT(w, coords + half2(1, 0));
  half4 wB = h4texRECT(w, coords - half2(0, 1));
  half4 wT = h4texRECT(w, coords + half2(0, 1));

  div = halfrdx * ((wR.x - wL.x) + (wT.y - wB.y));
}
```

Листинг 4. *Фрагментная программа вычитания градиента*

```
void gradient(half2 coords   : WPOS,   // координаты на сетке
   out half4 uNew : COLOR,  // новая скорость
   uniform half halfrdx,    // 0.5 / шаг сетки
   uniform samplerRECT p,   // поле давления
   uniform samplerRECT w)   // поле скорости
{
  half pL = h1texRECT(p, coords - half2(1, 0));
  half pR = h1texRECT(p, coords + half2(1, 0));
  half pB = h1texRECT(p, coords - half2(0, 1));
  half pT = h1texRECT(p, coords + half2(0, 1));

  uNew = h4texRECT(w, coords);
  uNew.xy -= halfrdx * half2(pR - pL, pT - pB);
}
```

Граничные условия. В пункте 2.4 определено, что наш флюид находится в «ящике» и граничные условия задаются равными нулю для скорости и граничные условия Неймана для давления. В пункте 3.2 показано, что граничные условия реализуются по периметру шириной в один пиксель сетки. Граничные условия обновляются при рисовании отрезков, используя фрагментную программу для соответствующих данных.

Сначала найдем, как данная дискретизация сетки влияет на вычисление граничных условий. Из условия отсутствия протекания на границе следует, что скорость равняется нулю на границе, а условие Неймана для давления требует равенства нулю производной давления по нормали к границе. Граница проходит между граничной ячейкой и ближайшей внутренней ячейкой, но значения на сетки определены в центрах ячеек. Следовательно, необходимо вычислять граничные значения так, чтобы среднее между значениями в двух смежных ячейках к границе удовлетворяло граничному условию.

Например, для скорости на левой границе имеем

$$\frac{\mathbf{u}_{0,j} + \mathbf{u}_{1,j}}{2} = 0, \quad j \in [0, N], \tag{14}$$

где N — разрешение сетки. Для удовлетворения этого условия необходимо установить $\mathbf{u}_{0,j}$, равное $-\mathbf{u}_{1,j}$. Уравнение давления рассматривается аналогично.

Используя разностную аппроксимацию производной, получим

$$\frac{p_{1,j} - p_{0,j}}{\delta x} = 0. \tag{15}$$

Из решения этого уравнения для $p_{0,j}$ видно, что необходимо установить значение величины давления на границе, равной величине давления во внутренней смежной ячейке к границе.

Можно использовать простую фрагментную программу как для давления, так и для скорости на границе, смотри листинг 5.

Листинг 5. *Фрагментная программа для граничных условий*

```
void boundary(half2 coords : WPOS,    // координаты на сетке
   half2 offset : TEX1,    // смещение вдоль границы
   out half4 bv : COLOR,   // выходное значение
   uniform half scale,     // параметр масштабирования
   uniform samplerRECT x)  // поле значений
{
  bv = scale * h4texRECT(x, coords + offset);
}
```

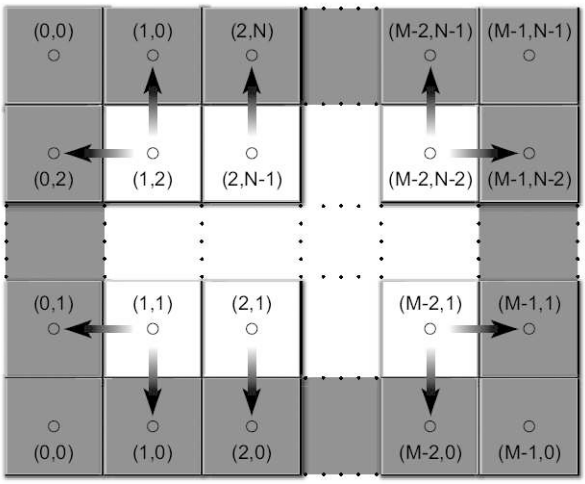

Рис. 5. Граничные условия на MxN-сетке

На рисунке 5 демонстрируется работа этой программы. Параметр x представлен текстурой (поле скорости или давления), из которой прочитывают внутренние величины. Параметр offset задает корректное смещение к внутренней ячейке, смежной с текущей граничной ячейкой. Параметр coords содержит позицию в координатах текстуры обрабатываемого фрагмента, так что, прибавляя смещение к нему, получаем соседний тексель. На каждой границе устанавливается свое смещение для регулирования координат текстуры для текселей, находящихся внутри от границы. Для левой границы установлено значение смещения $(1, 0)$, чтобы найти соседний тексель справа от границы; для нижней

границы значение смещения — $(0, 1)$; и так далее. Параметр scale может быть использован для масштабирования значения, которое переносится на границу. Для граничных условий для скорости масштабный коэффициент равен -1, а для давления 1, для того чтобы правильно реализовать уравнения (14) и (15) соответственно.

3.4. Три измерения

Представленная реализация моделирования флюидов — двумерна. Её легко расширить до трехмерной. Вид уравнений существенно не изменится, но их количество увеличится, поскольку вектор скорости $\mathbf{u} = (u, v, w)$ будет трехмерным. В связи с этим фрагментные программы должны быть переписаны. Например, в двумерном пространстве ячейка имеет четырех соседей, а в трехмерном — шесть.

Большое изменение касается представления векторных и скалярных полей. Можно воспользоваться трехмерными текстурами, но аппаратные средства не поддерживают трехмерные текстуры со значениями с плавающей точкой. Поэтому можно уложить трехмерную текстуру в двумерную. Например, сетка 32x32x32 ячейки укладывается мозаикой в двумерную текстуру 256x128 ячеек. Этот метод называют плоскими 3D текстурами, более подробно он описан в [5].

4. Примеры

Приведем различные результаты моделирования флюидов на CPU и GPU.

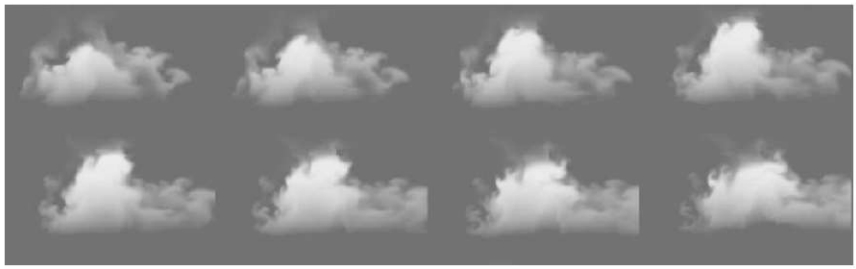

Рис. 6. Последовательность кадров моделирования облаков

Первый пример (рис. 6) представляет результат моделирования двумерного изображения облаков, реализованное М. Харрисом [7] на основе уравнения Навье-Стокса. Также в моделировании учитываются сила выталкивания и сила торможения турбулентности. Решение уравнений основано на декомпозиции Гельмгольца-Ходжа. На рис. 6 приведена последовательность кадров моделирования на сетке с разрешением 128x128 ячеек.

На рис. 7 приведен кадр имитации полета с отображением облаков. Облака генерируются в виде множества частиц различных форм и плотностей. Наложение изображений облаков обеспечивает вполне реальное воспроизведение объема. Подробнее можно ознакомиться в работе М. Харриса и А. Ластра [8].

Рис. 7. Имитация трехмерных облаков

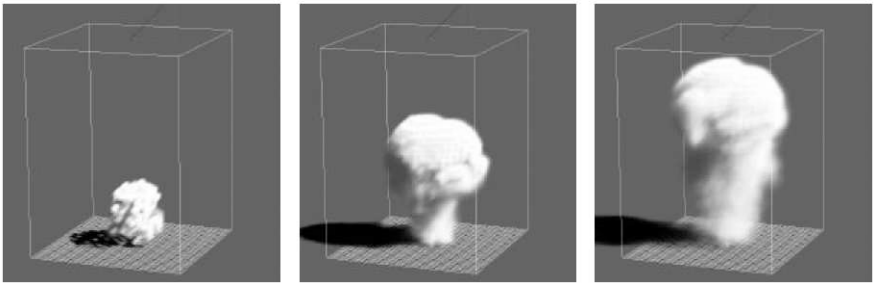

Рис. 8. Имитация распространения дыма

В работе Дж. Стама [12] эффективность метода устойчивых флюидов продемонстрирована на примере распространения дыма (рис. 8). Уравнения Навье–Стокса были реализованы на трехмерной сетке. Вычисления производились на CPU.

Дж. Крюгер и Р. Вестерманн в работе [9] демонстрируют модель двумерной поверхности воды как пример использования предложенного авторами алгоритма решения систем алгебраических уравнений на GPU (рис. 9).

В работе В. Ли, Ж. Фана, К. Вейя и А. Кауфмана [10] представлена модель потока флюида, основанная на сеточном методе Больцмана (Lattice Boltzmann Method, LBM). Рисунок 10 показывает результаты моделирования потока, обтекающего различные препятствия: неподвижная ваза, движущаяся сфера, плывущая медуза. Авторы модели отмечают, что при движении медуза деформирует свое тело, поэтому изменяется граница между жидкостью и телом медузы. Линиями на рисунке представлены направления движения потока. Все этапы

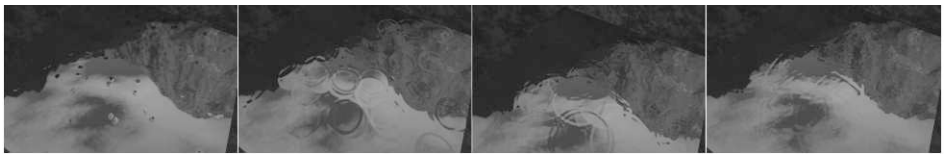

Рис. 9. Последовательность кадров моделирования поверхности воды

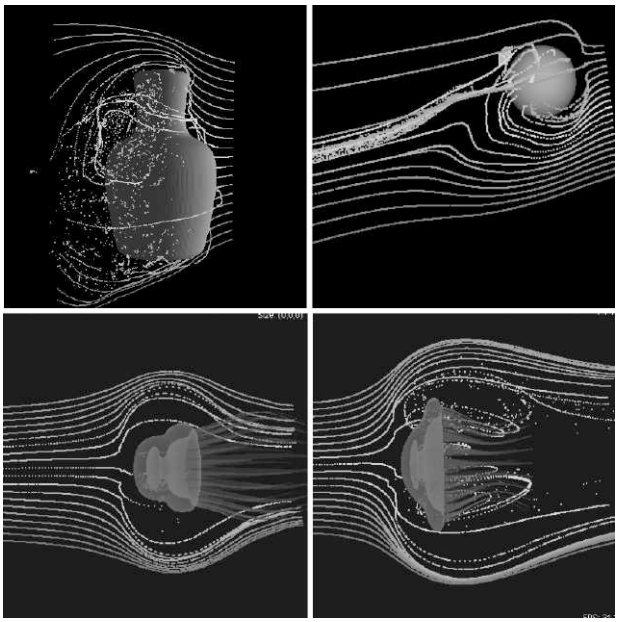

Рис. 10. Имитация поля потока обтекания предметов

вычисления и отображения выполнялись на GPU в реальном времени.

Литература

1. Bolz, J. Sparse Matrix Solvers on the GPU: Conjugate Gradients and Multigrid / J. Bolz, I. Farmer, E. Grinspun, P. Schröder // Proceedings of SIGGRAPH. – 2003. – P. 917–924.
2. Chorin, A.J. A Mathematical Introduction to Fluid Mechanics / A.J. Chorin, J.E. Marsden. – Springer-Verlag, 1993.
3. Golub, G.H. Matrix Computations / G.H. Golub, C.F. van Loan. – The Johns Hopkins University Press, 1996.
4. Goodnight, N. A Multigrid Solver for Boundary Value Problems Using Graphics Hardware / N. Goodnight, C. Woolley, G. Lewin, D. Luebke, G. Humphreys // Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware. – 2003. – P. 102–111.
5. Harris, M.J. Simulation of Cloud Dynamics on Graphics Hardware / M.J. Harris, W.V. Baxter, T. Scheuermann, A. Lastra // Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware. – 2003. P. 92–101.
6. Harris, M.J. Fast Fluid Dynamics Simulation on the GPU / M.J. Harris // GPU Gems, edt. Randima Fernando. – Addison-Wesley, 2004. – P. 637–665.
7. Harris, M.J. Real-Time Cloud Simulation and Rendering / M.J. Harris // University of North Carolina Technical Report TR03-040. – 2003.
8. Harris, M.J. Real–time cloud rendering / M.J. Harris, A. Lastra // Proceedings of Eurographics. – 2001. – P. 76–84.
9. Krüger, J. Linear Algebra Operators for GPU Implementation of Numerical Algorithms / J. Krüger, R. Westermann // Proceedings of SIGGRAPH. – 2003.

10. Li W. Flow Simulation with Complex Boundaries / W. Li, Z. Fan, X. Wei, A. Kaufman // GPU Gems II: Programming Techniques for High-Performance Graphics and General-Purpose Computation, edt. Matt Pharr. – Addison-Wesley, 2005. – P. 747–764.
11. Stam, J. Stable Fluids / J. Stam // Proceedings of SIGGRAPH. – 1999. – P. 121–128.
12. Stam, J. Real-time fluid dynamics for games / J. Stam // Proceedings of the Game Developers Conference. – 2003.