

Универсальные вычисления на графическом процессоре с использованием OpenGL ES 2.0



Владимир Черницкий
Азофт
2012



Эбру - искусство рисования красками на поверхности воды.

CPU



Множественный поток команд и данных (MIMD)

GPU



Одиночный поток команд и множество потоков данных (SIMD)

- Анализ и обработка изображений
(различного рода фильтрации изображений,
алгоритмы распознавания образов)
- Обработка звука и видео
- Дополненная реальность
- Вычислительная математика
- Динамика газов и жидкостей
- Криптография
- Геоинформационные системы
- Компьютерное зрение
и многое другое



Универсальные
вычисления
на графическом
процессоре

GPGPU

(General Purpose Computing
on Graphics Processing Units)

Особенности реализации (ограничения) GPU мобильных устройств

Ограниченные возможности
энергопотребления
и отвода тепла,
желание производителей
получить конкурентную
стоимость



Невысокая пропускная
способность памяти
между CPU и GPU

Сниженные частоты
ядер и памяти

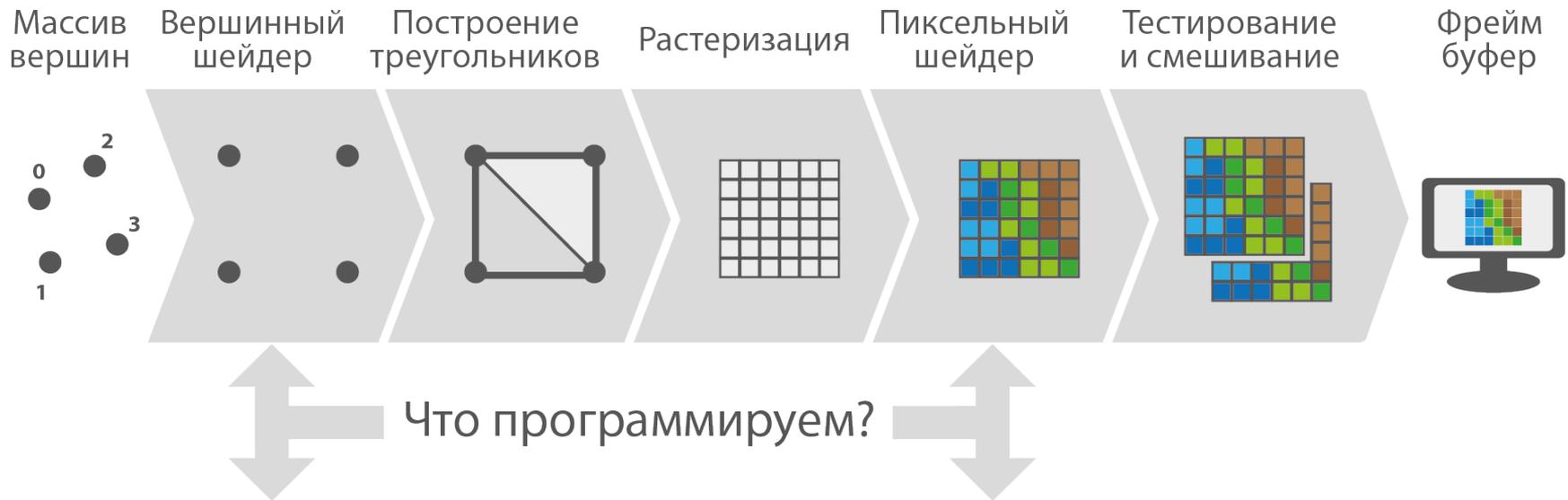
Архитектурные ограничения
(отсутствие поддержки float текстур,
GPU последних моделей поддерживают
16bit float текстуры, ограничения на
количество обрабатываемых буферов,
текстур и некоторые другие)

Отсутствие инструментария
для разработки GPGPU
~~OpenCL, Nvidia Cuda, Amd Stream,~~



OpenGL ES 2.0

Графический программируемый конвейер OpenGL 2.0



Вершинный шейдер

Графический процессор читает каждую вершину и применяет к ней вершинный шейдер. Основная задача этого шейдера - позиционирование вершины в 3D-пространстве

1000 вершин инициируют 1000 вызовов вершинного шейдера

Пиксельный шейдер

Все пиксели, полученные в процессе растеризации, проходят через пиксельный шейдер. Основная задача этого шейдера - это накладывание текстур, расчет освещенности и других визуальных эффектов.

Текстура 256x256 инициирует 65536 вызовов пиксельного шейдера

На чем программируем?

Шейдерный язык OpenGL носит название GLSL (The OpenGL Shading Language). GLSL основан на языке ANSI C. Большинство возможностей языка ANSI C сохранено, к ним добавлены векторные и матричные типы данных, часто применяющиеся при работе с трехмерной графикой

Переменные

Attribute

Varying

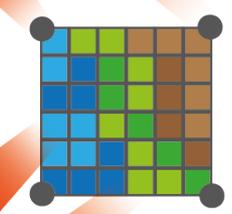
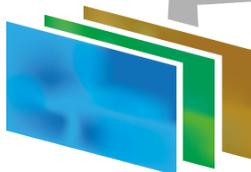
Uniform

Вершинный шейдер

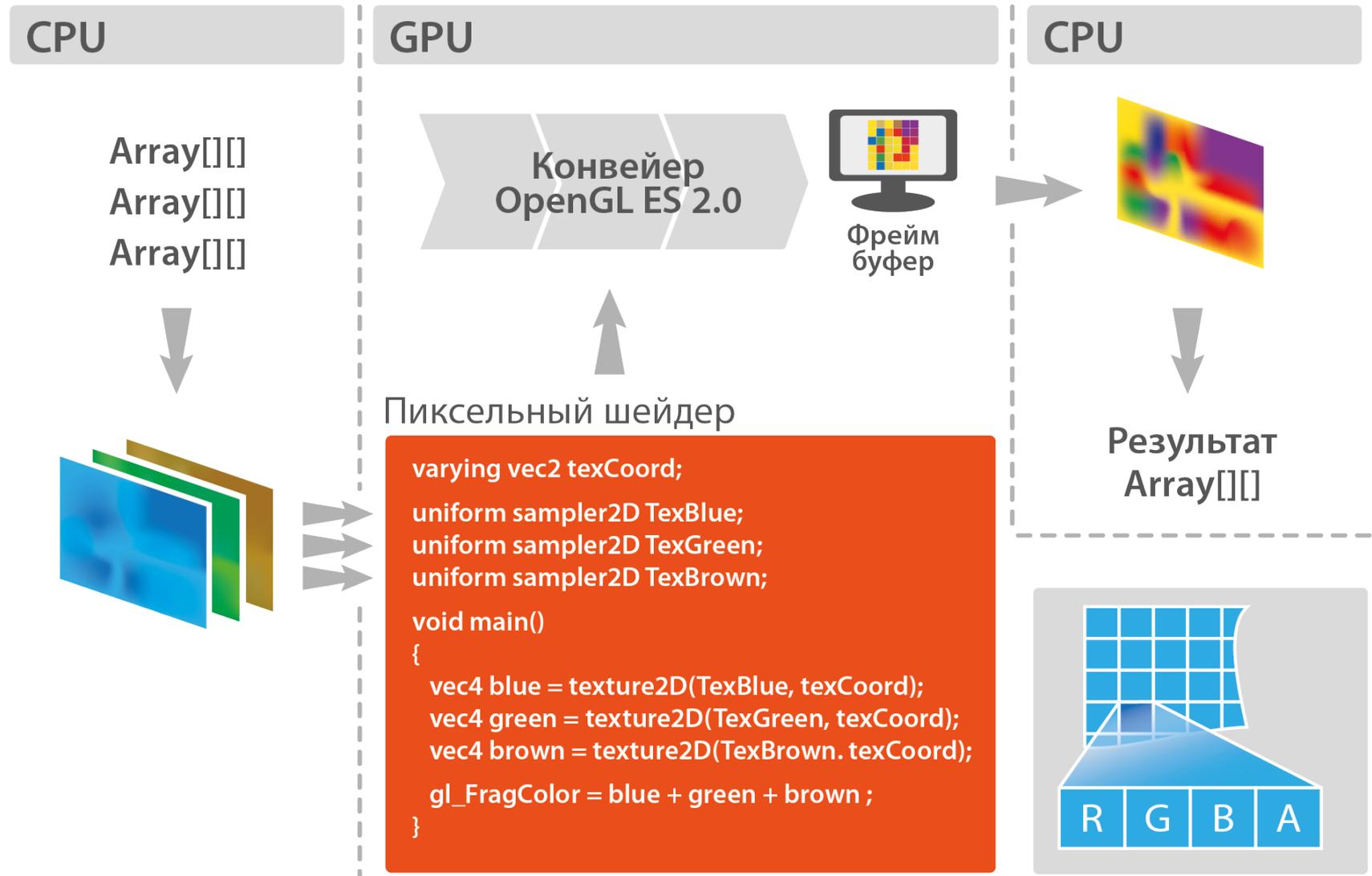
```
attribute vec4 inputPosition;  
attribute vec2 inputTextureCoordinate;  
  
varying vec2 textureCoordinate;  
  
void main()  
{  
    gl_Position = position;  
    textureCoordinate = inputTextureCoordinate.xy;  
}
```

Пиксельный шейдер

```
varying vec2 textureCoordinate;  
uniform sampler2D Texture;  
  
void main()  
{  
    gl_FragColor = texture2D(Texture, textureCoordinate);  
}
```



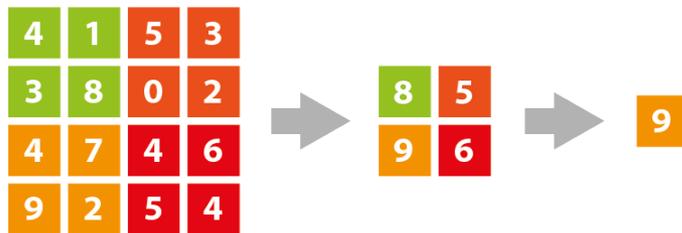
Простейшая схема приложения GPGPU



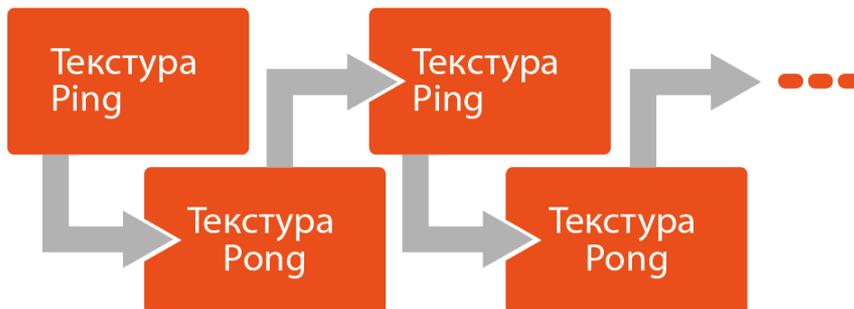
Некоторые техники, используемые при программировании GPGPU

Просто и быстро

- Мэппинг (Map operation)
- Чтение из памяти (Getter)
 $x = a[i]$
- Уменьшение потока (Reducing)



- Игра в Ping-Pong (Playing Ping-Pong)

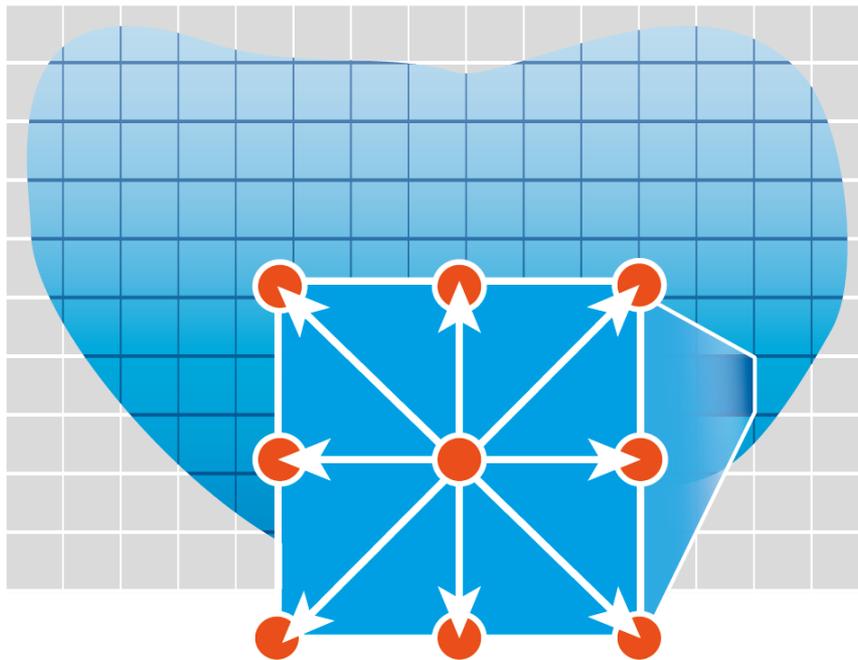


Не просто и не быстро

- Запись в память по произвольному адресу (Setter)
 $a[i] = x$
- Фильтрация
- Сортировка

Решеточный метод Больцмана (Lattice Boltzmann method)

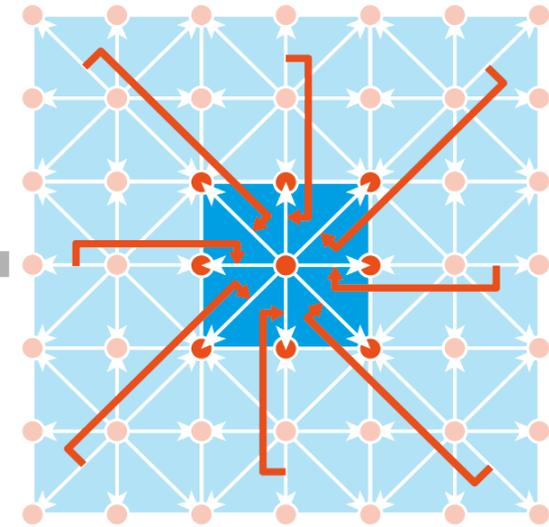
Представим жидкость, как однородную прямоугольную решетку



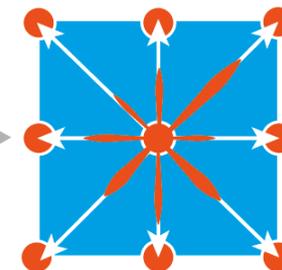
Модель D2Q9 (двухмерная, 9 скоростных каналов)

Эволюция решетки

Фаза переноса



Фаза столкновений



Блок-схема алгоритма

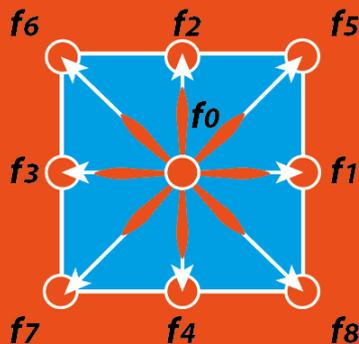
Иницилируем исходные гидродинамические переменные

$\rho = 1.0;$
 $U_x = 0.0;$
 $U_y = 0.0;$



Расчитываем плотность частиц в каждом скоростном канале (равновесное распределение)

$$f_i^{eq}(\rho_{xy}, U_{xy})$$



Техника Ping-Pong (x2)

Основной цикл

Внешние воздействия



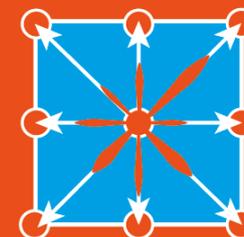
U_x, U_y
 uniform
 переменные
 x, y

0,0	0,0	0,0	0,0	0,0
0,0	0,0	0,0	0,0	0,0
0,0	0,0	4,-4	0,0	0,0
0,0	0,0	0,0	0,0	0,0

Фаза столкновений

$$f_i += relaxCoef * (f_i^{eq} - f_i)$$

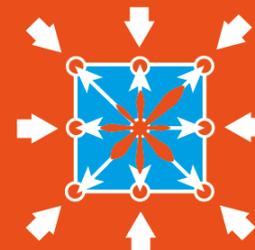
x3



Фаза переноса

Копируем в нашу ячейку значения соответствующих каналов соседних ячеек

x3



Рассчитываем обновленные значения гидродинамических параметров

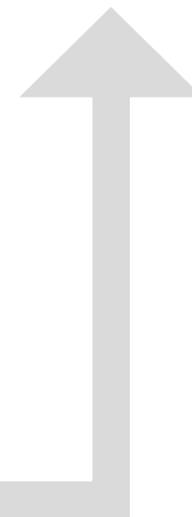
```
rho =  $\sum f_i$ ;  
Ux = 1/rho * (f5 + f1 + f8 - f6 - f3 - f7);  
Uy = 1/rho * (f6 + f2 + f5 - f7 - f4 - f8);
```

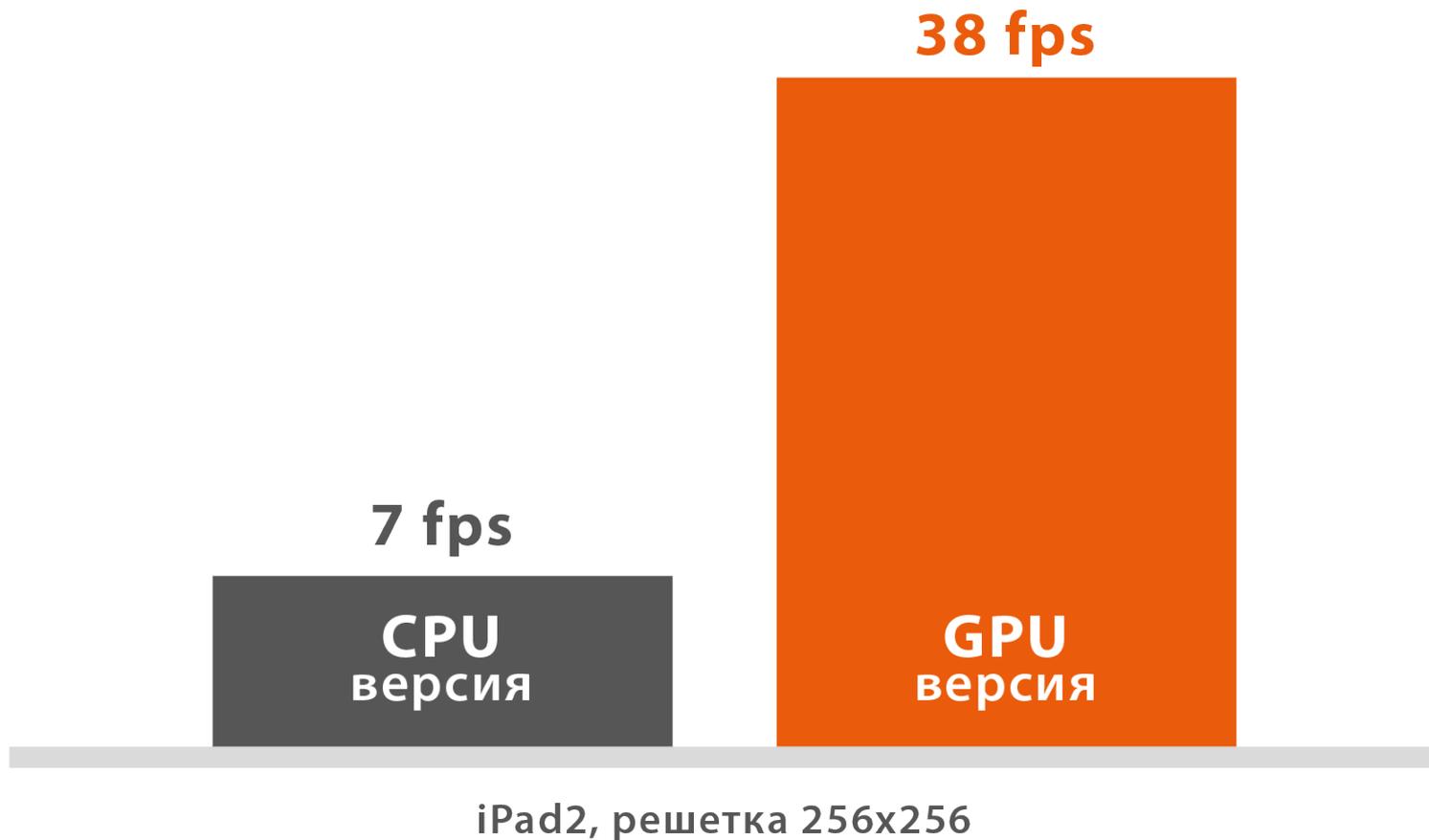
Визуализация

```
Записываем в красный канал  
длину вектора скорости в ячейке  
gl_FragColor = vec4(length(Ux, Uy), 0, 0, 1);
```



Возвращаемся в фазу
внешних воздействий





Полученные результаты показывают
значительное преимущество GPU
при выполнении хорошо распараллеливаемых вычислений



Для некоторых типов расчетов -
единственный способ получить
приемлемую производительность

Легкость портирования на другие
платформы с поддержкой
OpenGL ES 2.0



Громоздкость и недружелюбность,
сложность в отладке и тестировании

Ограниченные возможности

Нестандартный подход в реализации
алгоритмов расчетов

OpenCL специализированный фреймворк
для осуществления параллельных
вычислений

Свежие GPU уже имеют поддержку OpenCL на хардварном уровне

Следует ожидать появления поддержки стандарта
в ближайших SDK мобильных платформ

Спасибо за внимание

Азофт
Владимир Черницкий
skype: vladimir.tchernitski
amba@azoft.com