NVIDIA®

CUDA:
Новая архитектура для вычислений на GPU

Обзор

- Традиционная модель GPGPU
- Модель программирования CUDA
- Взаимодействие с графическими API
- Пример реализации гистограммы

Почему GPU?

- GPU является программируемым процессором:
  - С поддержкой языков высокого уровня
  - С поддержкой 32-битной плавающей точкой IEEE-754
  - Большой вычислительной мощностью:

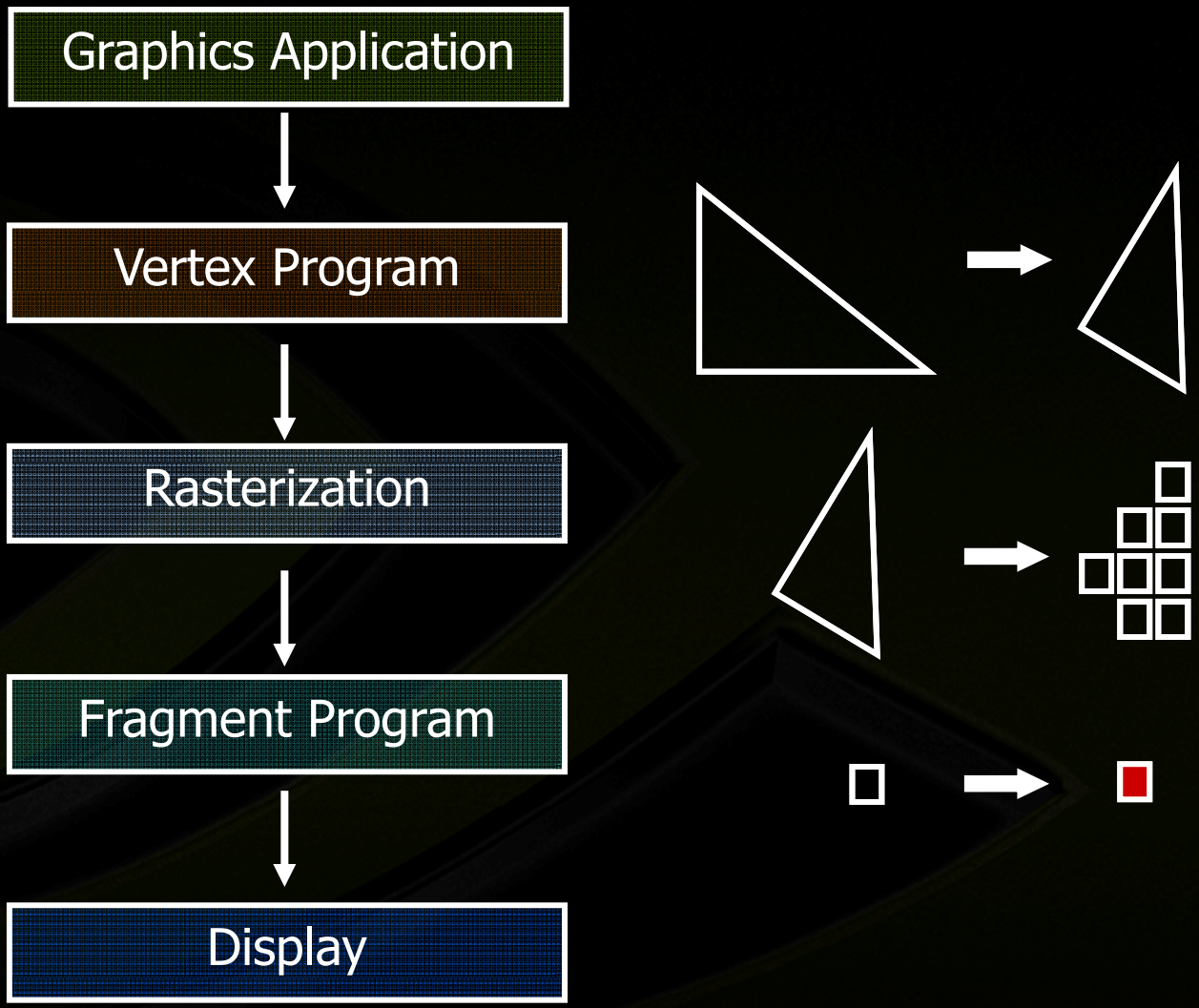
G80GL = Quadro FX 5600
G80 = GeForce 8800 GTX
G71 = GeForce 7900 GTX
G70 = GeForce 7800 GTX
NV40 = GeForce 6800 Ultra
NV35 = GeForce FX 5950 Ultra
NV30 = GeForce FX 5800

- Уже давно в компьютере каждого геймера!

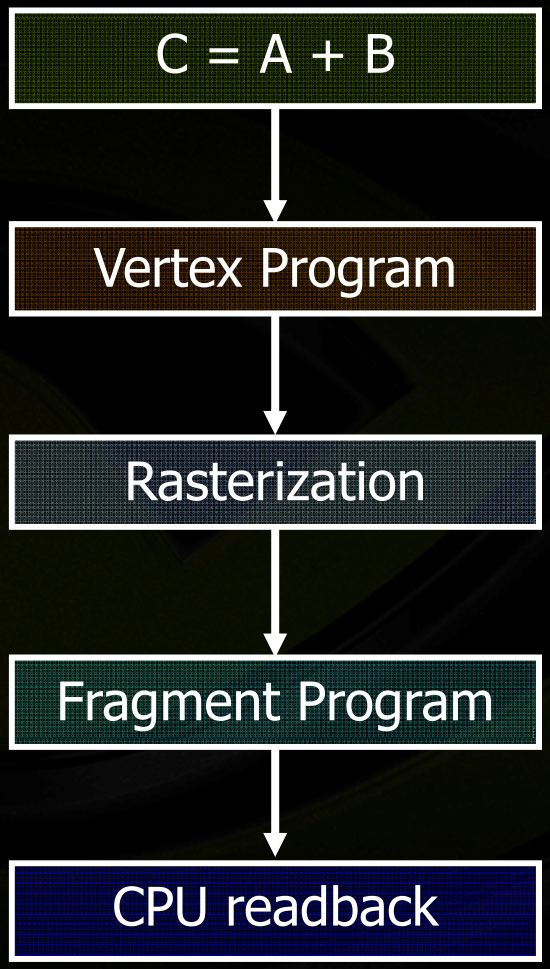
История GPGPU

- Графические процессоры (GPUs) использовались для неграфических вычислений в течение нескольких лет
- General-Purpose computation on GPUs: GPGPU
- Приложения GPGPU:
  - Симуляция физики
  - Обработка сигналов
  - Вычислительная математика/геометрия
  - Операции с базами данных
  - Вычислительная биология
  - Вычислительная экономика
  - Компьютерное зрение

Модель программирования GPU

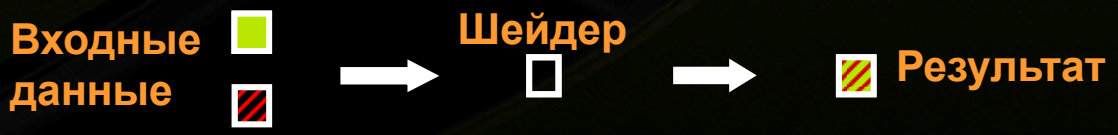


Традиционная модель программирования GPGPU



Рисуем quad

“Потоки” запускаются при растеризации



Традиционная модель GPGPU

Ограничения традиционной модели GPGPU

Что не так?

- *GPGPU* так или иначе -- трюк
  - Данные → изображения (“текстуры”)
  - Алгоритм → синтез изображения (“растеризация”)
- Результаты обещающие, но:
  - Обучение требует значительных ресурсов
  - Потенциальная субоптимальность графических API
  - Слишком специфическая модель памяти и исполнения

Решение: CUDA

CUDA = *Compute Unified Driver Architecture*

- Специализированный программно-аппаратный стек для вычислений на GPU
- Полная аппаратная поддержка на GeForce 8*, Quadro FX 5600/4600, Tesla и выше
- Не порываем с графикой!

CUDA: программная среда

- Пишем для GPU на настоящем C
  - Стандартный синтаксис
    - Указатели и не только…
  - Минимальные расширения для доступа ко всем вычислительным ресурсам аппаратуры

CUDA: аппаратура

- GPU –архитектура общего назначения для задач с параллелизмом данных
  - Скалярные RISC инструкции
  - Все типы математических операций
  - Глобальная память
  - Разделяемая память

Пиксель

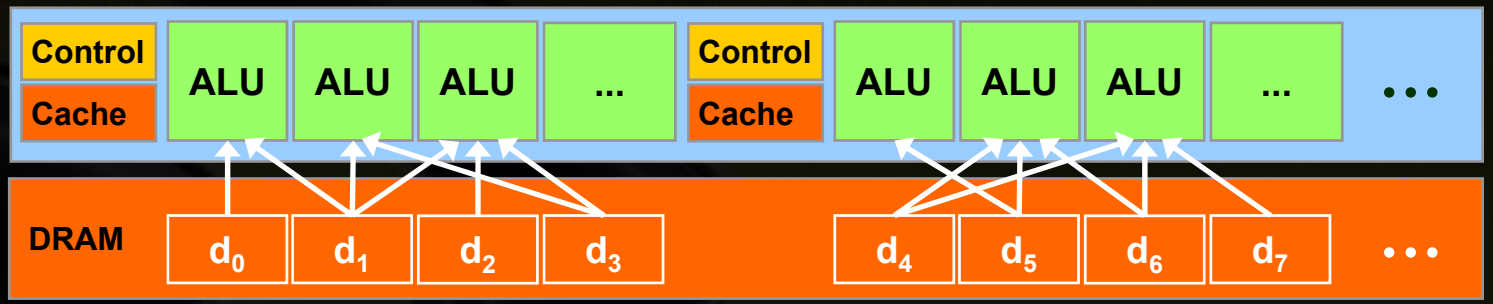
CUDA: функциональный поток

CUDA: глобальная память

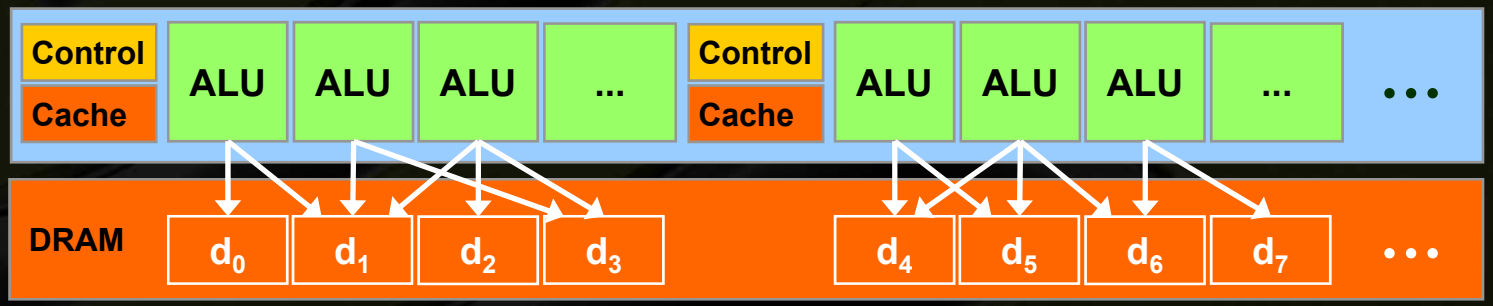
- Обобщённые инструкции Load/Store
- Обычные указатели на память

Преимущества глобальной памяти

- Как gather:



- Так и scatter:



→ Большая гибкость

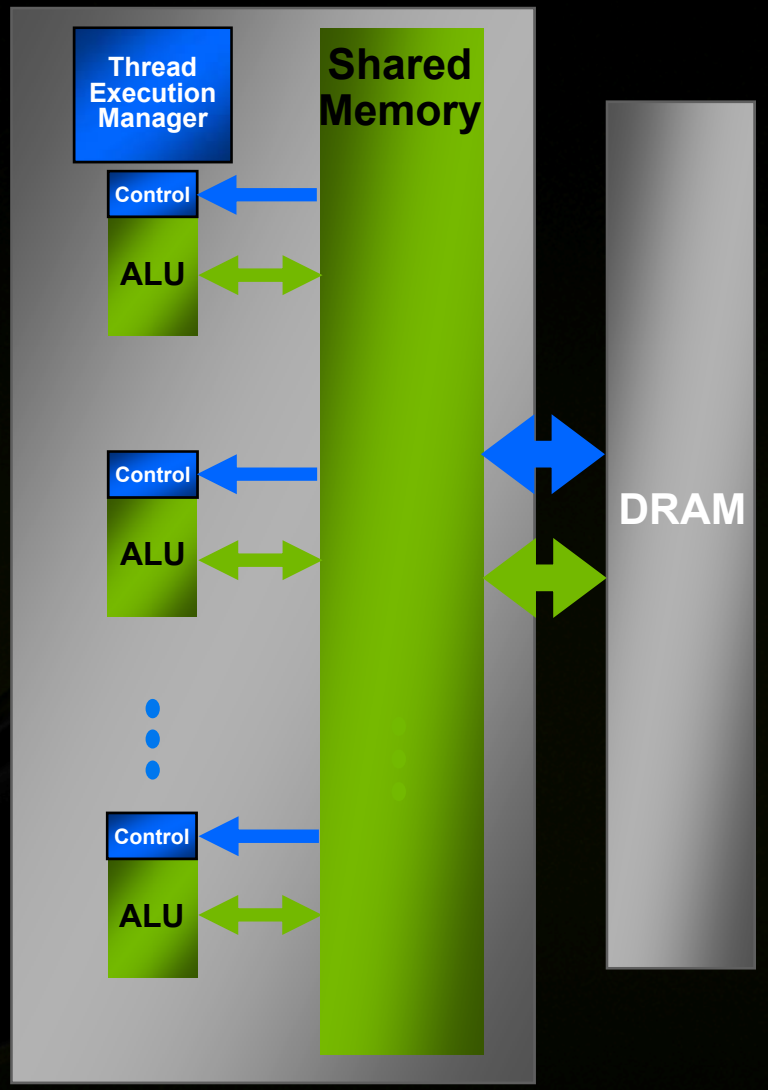
CUDA: разделяемая память

Преимущества разделяемой памяти

Уменьшает расстояние между АЛУ и данными

- Сокращая число обращений к внешней памяти
- Минимизируя избыточность загрузки и вычислений

Управляемый программистом L1-кэш



Модель программирования: Массивно-параллельный процессор

- GPU – вычислительное устройство:
  - Сопроцессор (device) для центрального процессора (CPU / host).
  - С собственной памятью DRAM (память устройства, device memory)
  - Параллельно обрабатывающее множество потоков исполнения
- Ядро (kernel) – функция для GPU, исполняемая исполняемая большим количеством потоков.
  - Графическая аналогия – шейдер
- GPU способен держать тысячи потоков в обработке
  - Естественно для мультимедиа
  - Нет избыточности создания/уничтожения

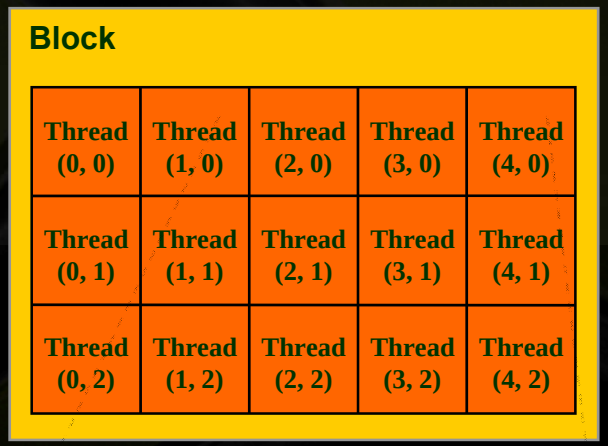
Модель программирования: поток GPU

- Скалярен
  - Нет необходимости упаковывать данные в 4-х векторы
  - Естественно для большинства задач
  - Меньше работы для компилятора
- Функциональная единица
  - В отличие от пикселей и вершин не является пассивным элементом данных!

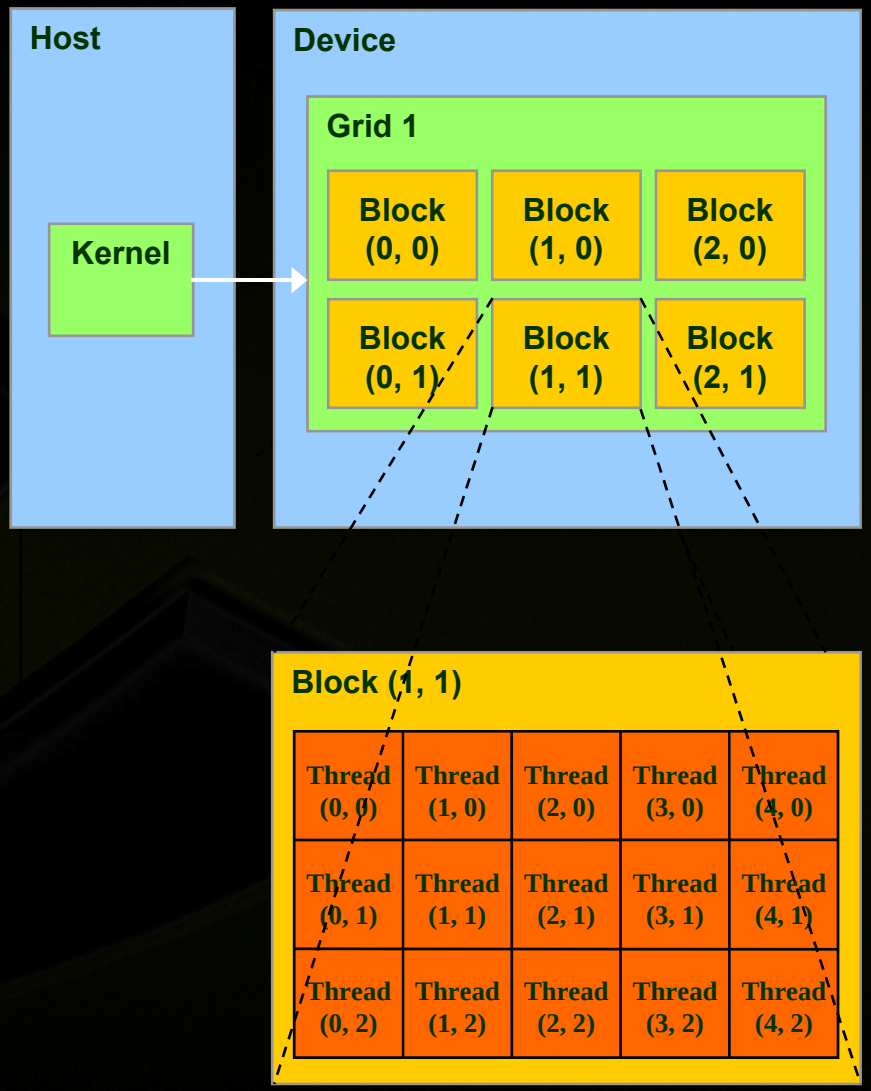
Модель программирования: блок потоков

Прямоугольная сетка потоков, способных **взаимодействовать** между собой посредством:

- **разделяемой памяти**
- точек синхронизации (**барьеров**)



Модель программирования: сетка блоков потоков



Модель программирования: масштабируемость

- Логических потоков и блоков потоков существенно больше, чем физических устройств
- Идеальная масштабируемость внутри модельного ряда GPU

C для GPU

- Дополнения только там, где нужно

```
__global__ void KernelFunc(...);

__shared__ int SharedVar;

KernelFunc<<<500, 128>>>(...);
```

- Семейство вызовов CUDA API делится на CPU-компоненту и GPU-компоненту.

C для GPU: встроенные переменные

- `uint3 gridDim;`
  - Размерности сетки блоков (в блоках)
- `uint3 blockDim;`
  - Размеры блока потоков (в скалярных потоках)
- `uint3 blockIdx;`
  - Индекс блока внутри сетки
- `uint3 threadIdx`
  - Индекс потока внутри блока
- Упрощает адресацию данных в ядрах (kernels) CUDA

C для GPU: функция синхронизации блока потоков


```
void __syncthreads();
```

- Синхронизует все потоки внутри блока потоков
- Обычно используется для предотвращения ошибок совместного доступа при работе с памятью
- Допускается внутри ветвлений только если все потоки блока гарантированно вычисляют одно и то же значение условия

C для GPU: пример запуска вычислений

```
//GPU-side
__global__ void KernelFunc(…){…}
//…
//CPU-side
dim3   DimGrid(100, 50);    //5000 блоков потоков
dim3   DimBlock(4, 8, 8);   //256 потоков на блок
KernelFunc<<<DimGrid, DimBlock>>>(...);
```

- Вызов ядра асинхронен
  - Управление немедленно возвращается к CPU

C для GPU: CPU-компонента библиотеки этапа исполнения

- Функции для:
  - Инициализации GPU (одного или нескольких)
  - Управления памятью GPU
  - Взаимодействия с OpenGL и Direct3D9
  - Обработки ошибок

Библиотека этапа запуска: управление памятью

- **Выделение** памяти
  - `cudaMalloc(), cudaFree(), ...`
- **Копирование** CPU→GPU, GPU→GPU, GPU→CPU
  - `cudaMemcpy(), ...`

Взаимодействие с графическими API

- CUDA работает на том же устройстве что и OpenGL / Direct3D!
- Ресурсы графических API могут быть отображены в пространство глобальной памяти CUDA
  - Для рендеринга данных, сгенерированных CUDA
  - Для считывания результатов рендеринга и дальнейшей обработки средствами CUDA

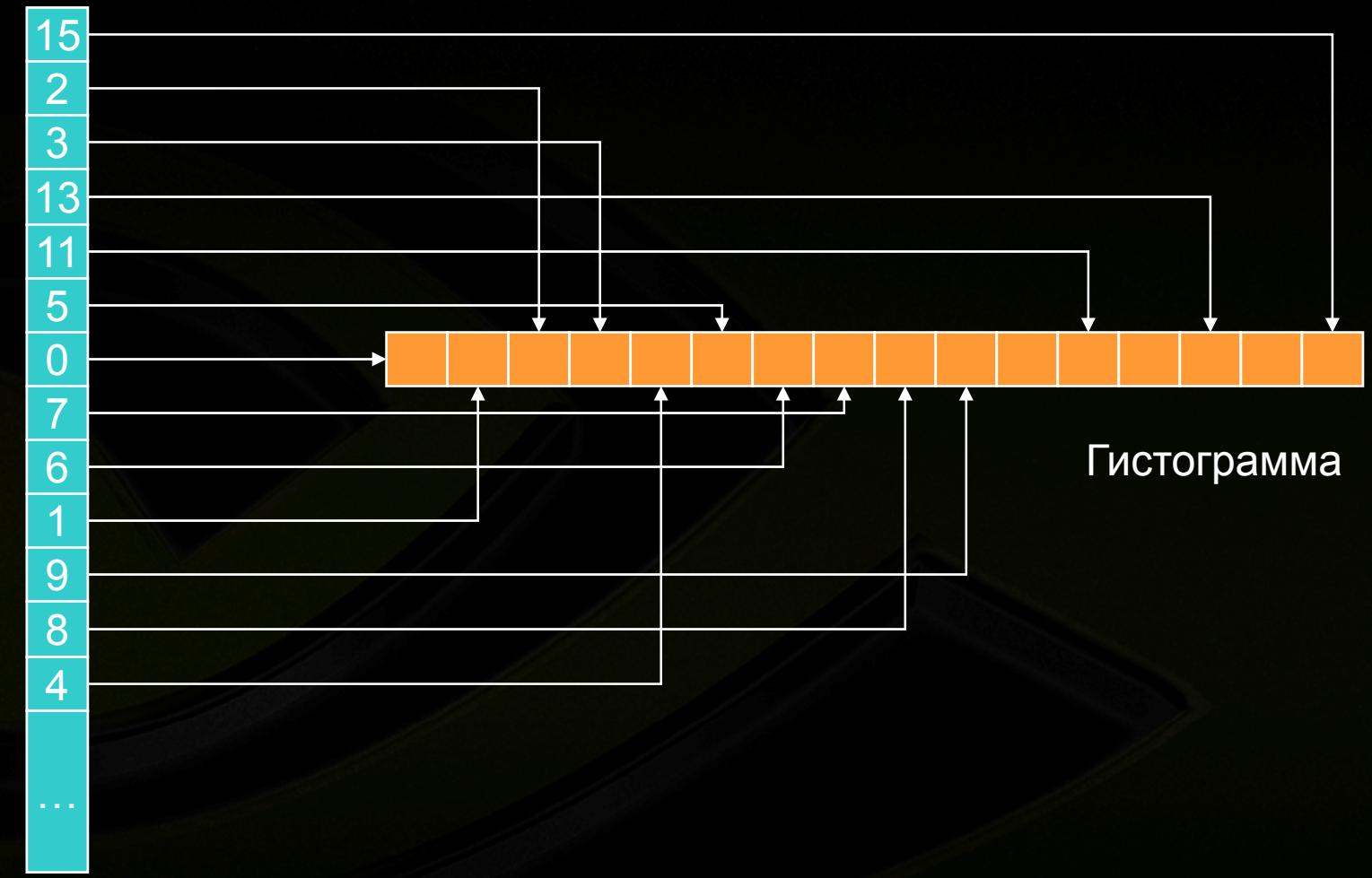
Взаимодействие с графическими API

- Отображение – получение адреса ресурса в глобальной памяти
  - `cudaGLMapBufferObject, …`
  - `cudaD3D9MapResources, cudaD3D9ResourceGetMappedPointer, cudaD3D9ResourceGetMappedSize, …`
- Поддерживаемые ресурсы
  - OpenGL Buffer Objects (PBO / VBO)
  - Direct3D9 Vertex Buffers
  - Текстуры Direct3D9 (2D / 3D / cubemap)

Применение CUDA в играх: вычисление гистограммы

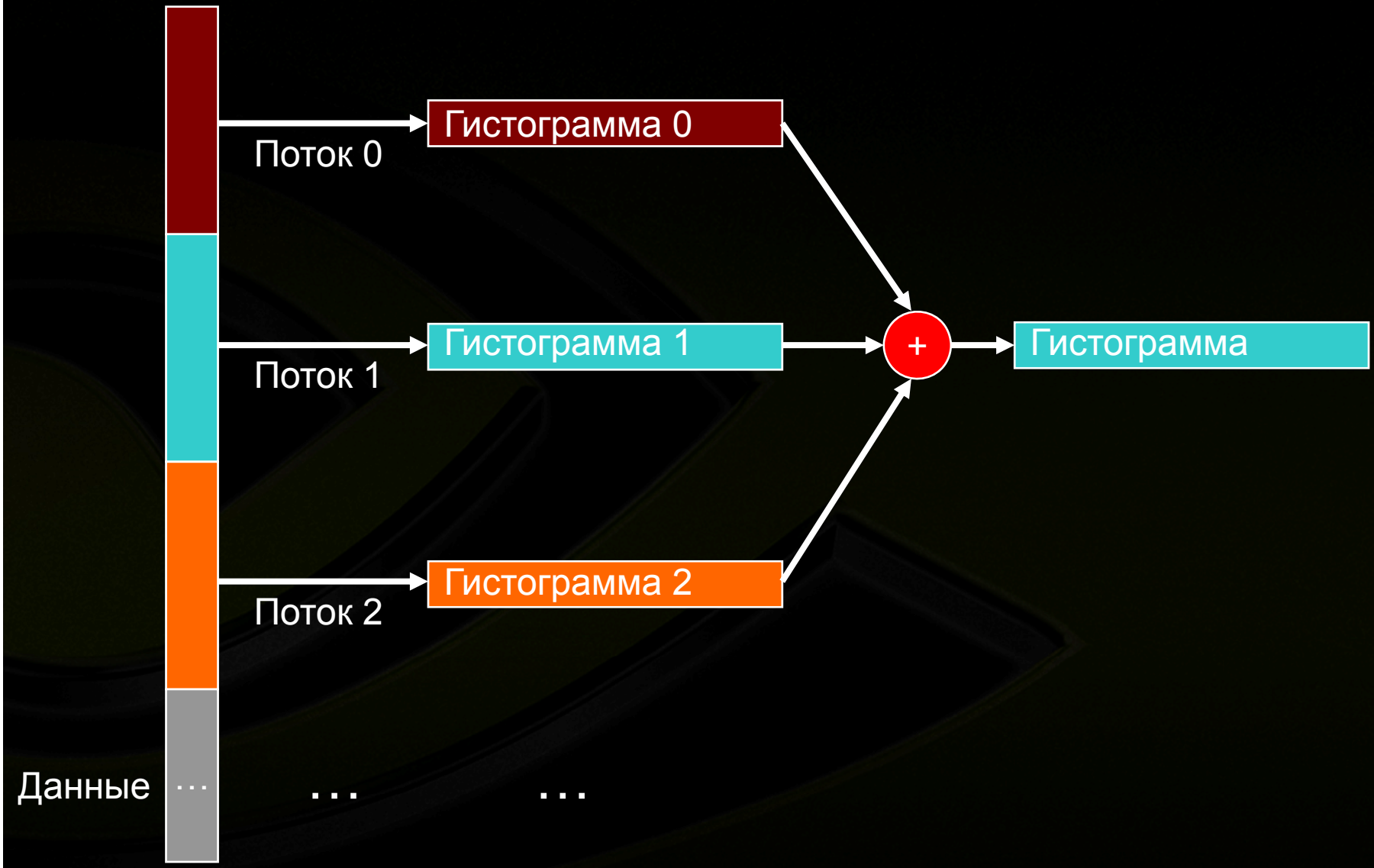
- Tonemapping – частый эффект в современных играх
- Вычисление гистограммы изображения – основа “честного” tonemapping
  - Пример: Half-Life 2 Engine
- Реализация средствами графических API очень трудоёмка
  - Приближённые вычисления очень низкого разрешения (Half-Life2 – 16 бинов)

Вычисление гистограммы в одном потоке исполнения

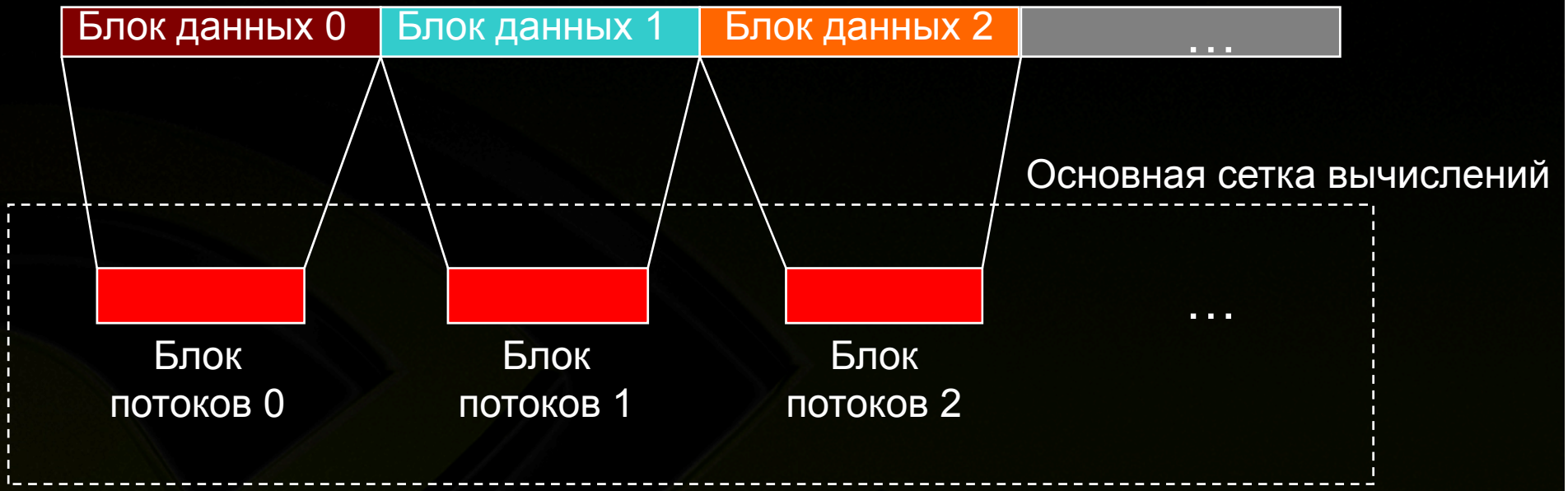


Параллелизация вычислений

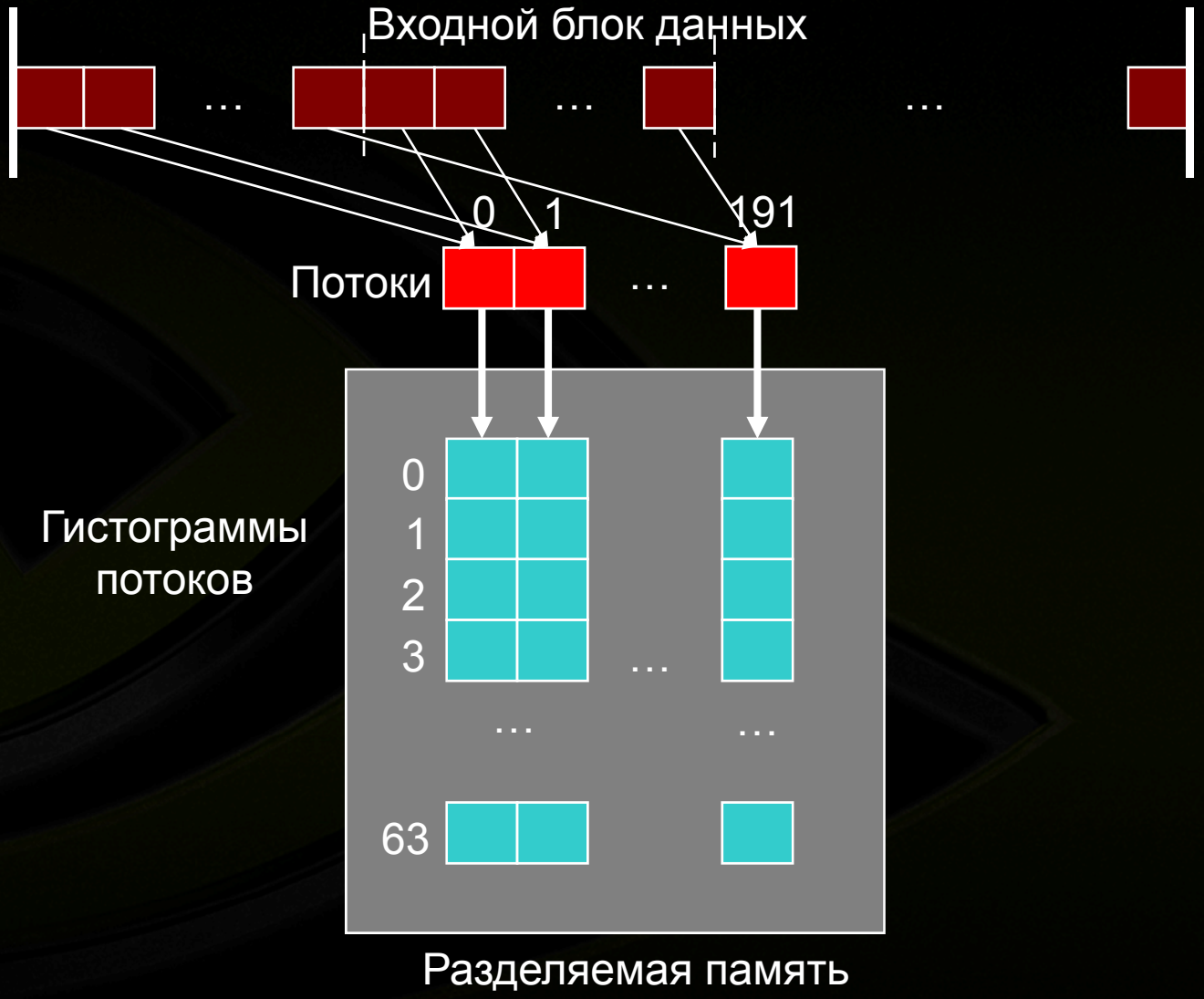
Объединение гистограмм



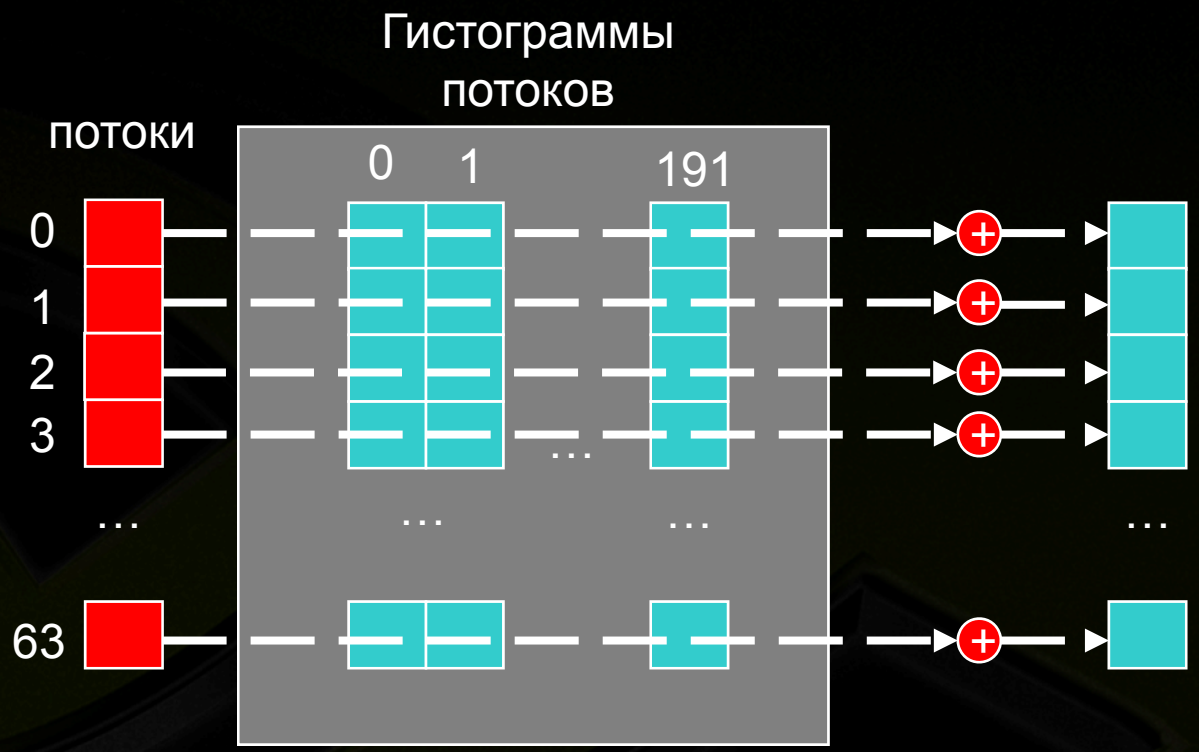
Гистограмма на CUDA: разбиение данных



Гистограмма на CUDA: вычисления внутри блока потоков



Гистограмма на CUDA: объединение гистограмм потоков



Гистограмма на CUDA: объединение гистограмм блоков потоков

Производительность

- Гистограмма на 64 бина (histogram64)
  - GeForce 8800 GTX: 10 Гпикс/с
- Гистограмма на 256 бина (histogram256)
  - GeForce 8800 GTX: 5,5 Гпикс/с

Заключение

- Интерфейс NVIDIA CUDA уже доказал свою применимость и эффективность в очень многих задачах.
- Совместное использование CUDA и графических API очень просто и приносит плоды
- Изучайте возможности GPU – основного элемента компьютеров Ваших пользователей!
- Начинайте экспериментировать сегодня!

Ссылки

- Сайт поддержки разработчиков для GPU
  - http://developer.nvidia.com/
- NVIDIA CUDA
  - http://www.nvidia.com/object/cuda_get.html
- Полная реализация histogram64 (CUDA SDK)
  - http://developer.download.nvidia.com/compute/cuda/1_1/Projects/histogram64.zip
- Полная реализация histogram256 (CUDA SDK)
  - http://developer.download.nvidia.com/compute/cuda/1_1/Projects/histogram256.zip