

ОБРАБОТКА ИЗОБРАЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ АППАРАТНОГО УСКОРИТЕЛЯ ГРАФИКИ



Разработчики: Дейнега В.М.

Малых Д.А.

Круглов В.Н.

Василий М. Дейнега,
Денис А. Малых, Р-54021

Исполнители

Научный руководитель:

Круглов Василий Николаевич, доцент кафедры АСУ,
Радиотехнический институт – РтФ, ГОУ ВПО «УГТУ – УПИ имени
первого президента России Б. Н. Ельцина».

Разработчики:

1. *Малых Денис Александрович*, студент Радиотехнического института – РтФ; разработчик систем искусственного интеллекта ООО «АйсХилл»; руководитель проекта iLLi Studio.
2. *Дейнега Василий Михайлович*, студент Радиотехнического института – РтФ; разработчик систем создания компьютерных игр ООО «Таргем Геймс»; технический руководитель проекта iLLi Studio.



Содержание

Часть 1: Реализация алгоритмов сглаживания и медианной фильтрации на аппаратных ускорителях графики

Часть 2: Framework обработки изображений на аппаратных ускорителях графики



Часть 1: Реализация алгоритмов сглаживания и медианной фильтрации на аппаратных ускорителях графики

- В современном мире большого прогресса достигли производители аппаратных ускорителей графики, в разработке систем визуализации компьютерных игр.
- Мощность слабой видеокарты многократно больше, чем мощность самого мощного центрального процессора персонального компьютера.
- Данная работа позволяет использовать потенциал современных графических систем в математических расчетах, не связанных с компьютерной графикой реального времени.
- В данной работе разработан способ использования GPU (Graphics Processing Unit) для реализации алгоритмов сглаживания и медианной фильтрации (классических алгоритмов обработки изображения).
- По результатам работы проведен сравнительный анализ со временем вычисления данных алгоритмов, в одном потоке на CPU.



Почему GPU?

Преимущества GPU:

1. Распараллеливание вычислений
2. RISC, масштабируемая архитектура
3. Программируемый конвейер (технология шейдеров)
4. Функционально ориентированный процессор
5. Низкоуровневая реализация математической библиотеки
6. Большой объем памяти (до гигабайта и выше)
7. Большая скорость обмена с памятью (до 64 Гб / сек.)



HLSL

Языки программирования шейдеров:

1. низкоуровневый
2. высокоуровневый

Пиксельный шейдер – это микропрограмма обработки одного пикселя текстуры. Обработка пикселей текстуры может вестись параллельно – поэтому на пиксельный шейдер накладываются определенные условия:

HLSL – это язык программирования шейдеров, который является высокоуровневым языком программирования. Именно он используется для написания шейдеров.

1. На входе в пиксельный шейдер может быть несколько текстур и несколько константных значений.
2. На выходе пиксельного шейдера может быть ТОЛЬКО цвет обрабатываемого пикселя.
3. Модель шейдеров ограничивает количество ассемблерных команд пиксельного шейдера. Кроме того, она накладывает ограничение на тип используемых операций. Например в модели 2.0 нельзя использовать циклы.

языка. Он

используется в системе.

Для обработки

различных шейдеров.

Алгоритм сглаживания

Классическая реализация алгоритма сглаживания выглядит следующим образом:

1. Запускается цикл по всем пикселям картинки
2. Для каждого пикселя составляется окно сглаживания (упомянуто выше).
3. Внутри этого окна вычисляется сумма цветов каждого пикселя.

4. Вычисленная сумма делится на общее количество пикселей в окне, тем самым вычисляется среднее арифметическое цветов пикселей в окне.

Таким образом формула расчета выглядит следующим образом:

$$p'_{i,j} = \frac{1}{(2w+1)^2} \sum_{x=-w}^w \sum_{y=-w}^w p_{x,y}$$

Результат:

CPU – 968 ms

GPU – 10 ms

Включает в себя усреднение цвета пикселей.

Размер окна до 128x128

После этого больше выполняется алгоритм.

Следует отметить следующие вещи:

1. Алгоритм проходит последовательно по всем пикселям картинки. Последовательно в данном случае значит долго.
2. Когда окно сглаживания достаточно большое – происходит повторная проход по большому количеству пикселей, причем по нескольку раз. И чем больше окно – тем большее количество раз выбирается цвет пикселя.

Медианная фильтрация

Классический алгоритм медианной фильтрации выглядит следующим образом:

1. Вводится функция med , которая вычисляет медиану нескольких элементов. Медианой является срединный элемент в упорядоченном массиве вход функции.

2. Каждый пиксель обрабатывается в некотором окне медианной фильтрации. Таким образом, цвет выходной функции вычисляется следующим образом:

$$p'_{i,j} = med(p_{i-w,j-w}, p_{i-w+1,j-w}, \dots, p_{i+w,j-w}, p_{i-w,j}, p_{i-w+1,j}, \dots, p_{i+w,j}, p_{i-w,j+w}, p_{i-w+1,j+w}, \dots, p_{i+w,j+w})$$

СИСТЕМЫ.

Результат:

CPU – 1890 ms

GPU – 4 ms

итием алгоритма

р зрения реального

- Однако, после проведения дополнительных оптимизаций, мы добились ускорения работы алгоритма путем разбиения его на два прохода.
- Первый проход вычисляет медиану только по строкам, второй – только по столбцам.
- На финальный результат такое разбиение не влияет благодаря свойствам медианной фильтрации.
- Формула будет выглядеть следующим образом:

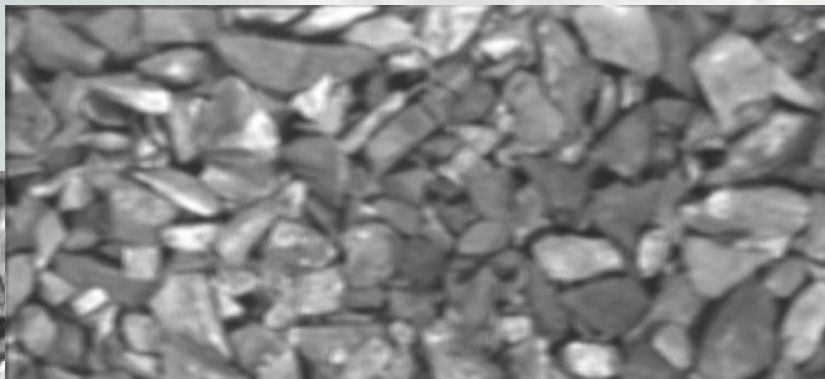
$$p'_{i,j} = med_x(med_y)$$

Примеры фильтрации:

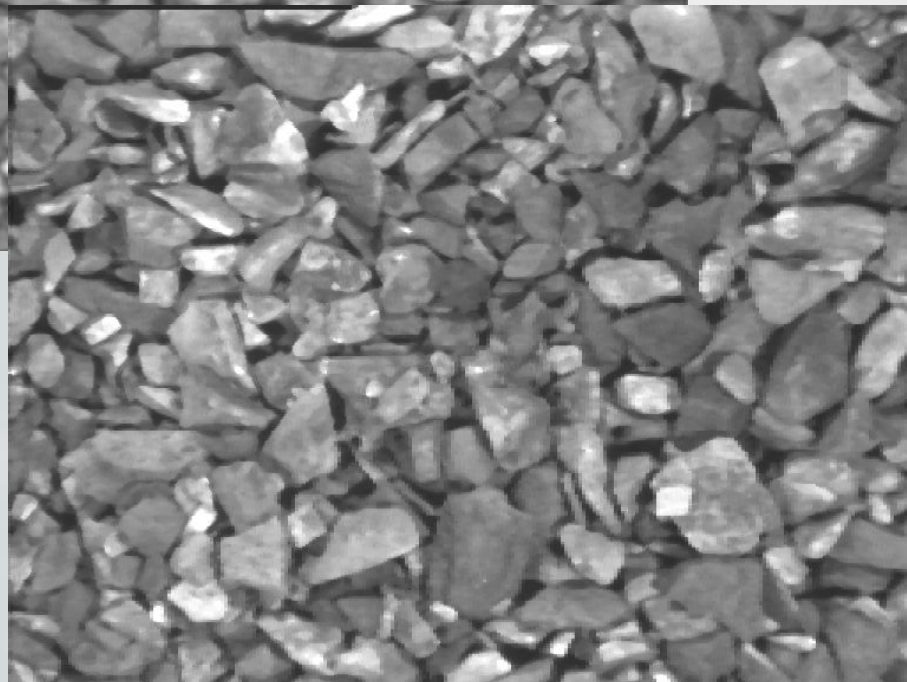
1



2



3



1. Исходное изображение
2. Blur
3. Медианная фильтрация

Часть 2: Framework обработки изображений на аппаратных ускорителях графики

- Иногда возникает необходимость повысить производительность математических вычислений.
- Один из кардинальных методов, был рассмотрен в части 1 данного доклада – это использование GPU вместо CPU.

Почему данные технологии не используются тогда повсеместно, и не ускоряют в тысячи раз математические расчеты во всех программах?

Ответ прост: Данные технологии применимы не ко всем типам алгоритмов и достаточно сложны в программировании и понимании.



Использование GPU

Когда стоит использовать GPU:

Если алгоритм можно представить, как множество одинаковых вычислений, производимых с разными данными.

Фреймворк в данной работе – это специальная библиотека, позволяющая создавать алгоритмы обработки изображений с использованием GPU, на уровне конструирования, а не программирования.

Ее можно подключить из любого языка, поддерживающего Windows программирование. Кроме того, эта библиотека существенно упрощает программирование инфраструктуры приложения и позволяет сосредоточиться на алгоритмах обработки изображений, а не на алгоритмах построения приложений.



Структура библиотеки

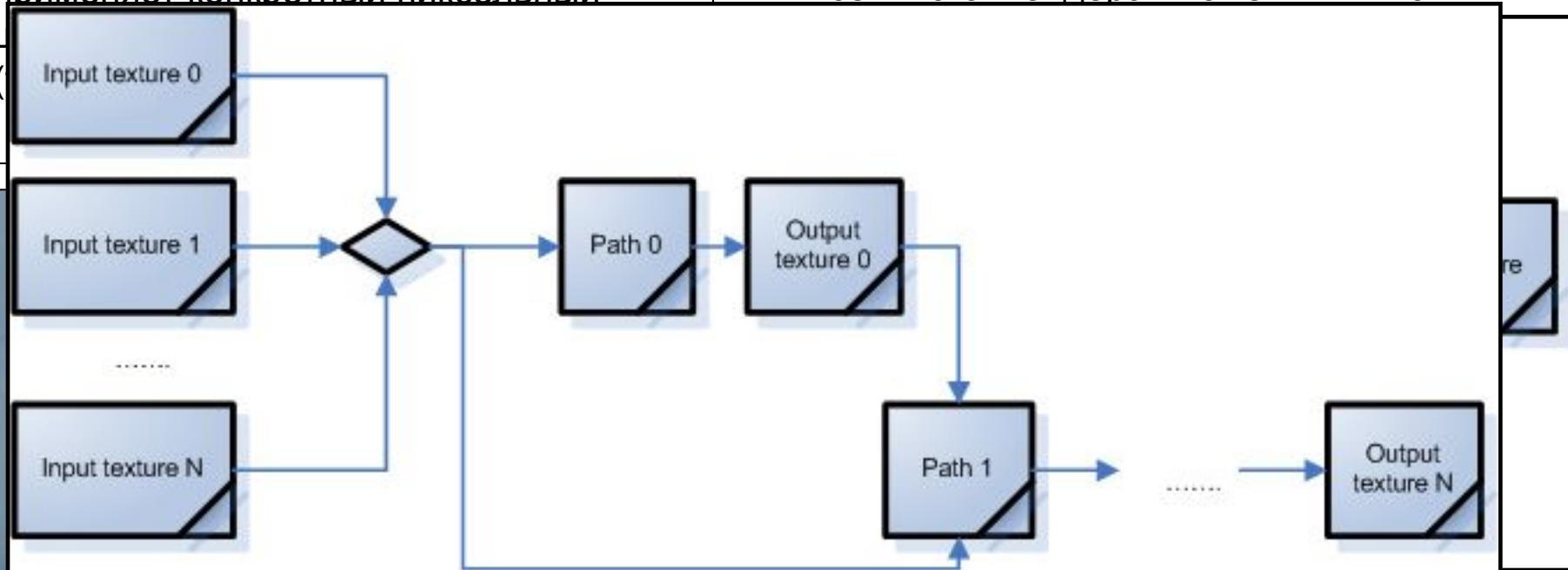
Текстура. Массив состоящий из 32

Пиксельный шейдер. Микропрограмма

Проход. Выполнение конкретного пиксельного шейдера к конкретным текстурами входа и выхода. Один раз применяет конкретный пиксельный

Алгоритм. Конкретный выполняемый

Рендер-таргет. От английского RenderTarget – выходная текстура обработки пиксельным шейдером. Текстура получаемая на выходе пиксельного шейдера. Может являться



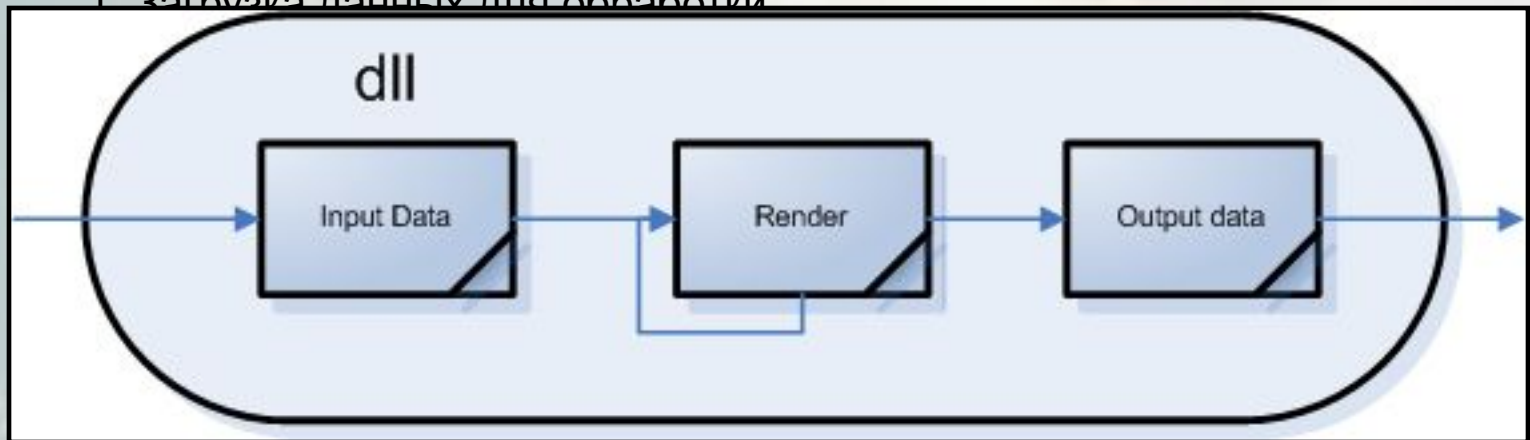
Конвейер обработки

Этап 1: Подготовка, загрузка данных

Этап 2: Описание используемых алгоритмов

Этап 3: Работа системы

1. Загрузка данных для обработки



2. Пакетная обработка данных (выполняется загрузка данных в систему, многократное выполнение разных алгоритмов, возврат данных).

3. Комбинированный метод.



Вопросы?

