**(слайд)**

Целью данной работы является создание демонстрационного приложения с использованием графического движка OptiX.

Задачи, решаемые в ходе работы:

1. Изучение программно-аппаратной архитектуры CUDA.

2. Изучение принципов функционирования графического движка OptiX.

3. Изучение процедуры установки графического движка OptiX.

4. Изучение встроенных примеров графического движка OptiX.

5. Разработка демонстрационного приложения.

6. Выяснение перспектив применимости графического движка в прикладных приложениях.

1 вопрос, который я задала, как работает движок?

Движок OptiX носит чрезвычайно общий характер, позволяя разработчикам программного обеспечения быстро ускорять выполнение любых задач на основе трассировки лучей и выполнять их на широко доступном оборудовании. Что же такое трассировка лучей?

**(слайд метод трассировки)**

Трассировка лучей (англ. Ray tracing; рейтрейсинг) --- один из методов геометрической оптики --- исследование оптических систем путём отслеживания взаимодействия отдельных лучей с поверхностями.

Данный метод имеет следующие достоинства:

1. Возможность рендеринга гладких объектов без аппроксимации их полигональными поверхностями (например, треугольниками).

2. Вычислительная сложность метода слабо зависит от сложности сцены.

3. Высокая алгоритмическая распараллеливаемость вычислений — можно параллельно и независимо трассировать два и более лучей, разделять участки (зоны экрана) для трассирования на разных узлах кластера и т.д.

4. Отсечение невидимых поверхностей, перспектива и корректное изменения поля зрения являются логическим следствием алгоритма.

2 вопрос, какая архитектура используется?

(слайд Cuda)

CUDA (Compute Unified Device Architecture) -- это технология от компании NVidia, предназначенная для разработки приложений для массивно-параллельных вычислительных устройств (в первую очередь для GPU).

GPU(графический процессор) — отдельное устройство [персонального компьютера](http://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D1%81%D0%BE%D0%BD%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80) или [игровой приставки](http://ru.wikipedia.org/wiki/%D0%98%D0%B3%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D0%BF%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%B2%D0%BA%D0%B0), выполняющее графический [рендеринг](http://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BD%D0%B4%D0%B5%D1%80%D0%B8%D0%BD%D0%B3" \o "Рендеринг).

**(слайд Развитие GPU)**

C самого начала GPU активно использовали параллельность (как вершины, так и отдельные фрагменты могут обрабатываться параллельно и независимо друг от друга, т.е. очень хорошо ложатся на параллельную архитектуру).

По мере развития GPU росла как степень распараллеливания, так и гибкость самих GPU. Самые первые GPU представляли собой просто растеризатор с возможностью наложения текстуры и буфером глубины. Довольно быстро появились GPU с полной обработкой вершин на самом GPU - на вход поступают трехмерные данные и на выходе получаем готовое изображение. Однако гибкость в них была небольшой - ведь все вычисления велись в рамках фиксированного конвейера.

Следующим шагом стало появления вершинных- обработку вершин стало возможным задавать в виде программы, написанной на специальном ассемблере.

Вполне логичным следующим шагом стало появление фрагментных, позволяющим задавать расчет каждого пиксела также при помощи программы, на ассемблере.

Заключительным шагом, превратившим GPU в мощные параллельные вычислители, стало поддержка floating-point текстур, т.е. стало возможным хранить значения в текстурах как 32-битовые floating-point числа.

В результате GPU фактически стало устройством, реализующим потоковую вычислительную модель (stream computing model) - есть потоки входных и выходных данных, состоящие из одинаковых элементов, которые могут быть обработаны независимо друг от друга. Обработка элементов осуществляется ядром (kernel).

**(слайд,рис потоковые вычисления)**

Появление CUDA предложила для GPGPU простую и удобную модель. В этой модели GPU рассматривается как специализированное вычислительное устройство (называемое device), которое:

Является сопроцессором к CPU (называемому host).

Обладает собственной памятью (DRAM).

Обладает возможностью параллельного выполнения огромного количества отдельных нитей (threads).

CUDA использует большое число отдельных нитей для вычислений, часто каждому вычисляемому элементами соответствует одна нить. Все нити группируются в иерархию - grid/block/thread

**(слайд рис.иерархия нитей)**

Верхний уровень - grid - соответствует ядру и объединяет все нити выполняющие данное ядро. grid представляет собой одномерный или двухмерный массив блоков (block). Каждый блок (block) представляет из себя одно/двух/трехмерный массив нитей (threads).

При этом каждый блок представляет собой полностью независимый набор взаимодействующих между собой нитей, нити из разных блоков не могут между собой взаимодействовать.

Подобная иерархия довольно естественна - с одной стороны хочется иметь возможность взаимодействия между отдельными нитями, а с другой - чем больше таких нитей, тем выше оказывается цена подобного взаимодействия.

3.вопрос принципы движка Optix.

**(слайд принципы Optix)**

Чтобы решить проблему создания доступной, гибкой, и эффективной системы трассировки лучей для массивно-параллельной архитектуры, представляем OptiX --- механизм трассировки лучей общего назначения.

Этот механизм комбинирует программируемый конвейер трассировки лучей с легким представлением сцены.

Чтобы создать систему для широкого диапазона задач трассировки лучей, были принято несколько компромиссов и проектных решений, которые привели к следующим особенностям:

**Общий низкоуровневый механизм трассировки лучей.**

Механизм OptiX фокусируется исключительно на фундаментальных вычислениях, требуемых для трассировки лучей, и избегает встраивания конструкции для рендеринга.

Движок представляет механизмы для выражения взаимодействий геометрии луча и не имеет встроенного понятия световых сигналов, теней, коэффициента отражения, и т.д.

**Программируемый конвейер трассировки лучей**.

Механизм OptiX демонстрирует, что большинство алгоритмов трассировки лучей могут быть реализованы, используя маленький набор легких программируемых операций.

Это определяет абстрактную модель выполнения трассировки лучей, поскольку последовательность пользователя определила программы.

Эта модель при объединении с произвольными данными, хранившимися с каждым лучом, может использоваться, чтобы реализовать множество сложных графических сцен и невизуальных алгоритмов.

Центральная идея механизма OptiX состоит в том, что большинство алгоритмов трассировки лучей может быть реализовано, используя маленький набор программируемых операций.

Это прямой аналог к программируемым конвейерам растеризации, используемым OpenGL и Direct3D.

На высоком уровне те системы представляют абстрактный растеризатор, содержащий легкие обратные вызовы для штриховки вершины, обработки геометрии, составления мозаики и операций штриховки фрагмента.

**Простая модель программирования.**

Механизм OptiX обеспечивает такие механизмы выполнения, что программисты пользуются знакомыми методами работы с трассировкой лучами и не обременяют себя низкоуровневой оптимизацией.

Движок представляет знакомую рекурсивную модель программирования единственного луча, а не пакеты луча или явные конструкции SIMD-стиля.

Есть семь различных типов программ в OptiX, каждая из которых работает над одним лучом одновременно.

Программа ограничительной рамки действует на геометрию, чтобы определить границы примитива для построения ускоряющей структуры. Программы генерации лучей являются начальной точкой в конвейере трассировки лучей. Программы ограничительной рамки вычисляют границы, связанные с каждым примитивом для включения ускоряющей структуры над произвольной геометрией. Программы ближайшего попадания вызываются один раз для обхода ближайшего пересечения луча с геометрией сцены. Программы любого попадания вызываются во время обхода для каждого пересечения объекта луча, который находится. Во время трассировки луча, вызванные программы попадания материала заполняют результирующее поле в определяемой пользователем структуре payload. Программы промаха выполняются, когда луч не пересекает любую геометрию в предоставленном интервале.Они могут быть использованы для реализации цвета фона или подстановки карты среды .

**Проблемно-ориентированный компилятор.**

Механизм OptiX комбинирует своевременные методы компиляции со специфичным для трассировки лучей знанием, чтобы реализовать его модель программирования эффективно.

Абстракция механизма разрешает компилятору настраивать модель выполнения для доступных системных аппаратных средств.

**Эффективное представление сцены.**

Механизм OptiX реализует объектную модель, которая использует динамическое наследование, чтобы упростить компактное представление параметров сцены.

Гибкая система графика узла позволяет сцене быть организованной для максимальной производительности, все еще поддерживая инстанцирование, многоуровневую детализацию и вложенные ускоряющие структуры.(слайд)

**Объектная модель**

OptiX использует объектную модель специального назначения, предназначенную для минимизации постоянных данных, используемых в программных операциях.

В отличие от системы OpenGL, где только одна комбинация шейдеров используется одновременно.

Тем не менее, трассировка лучей может случайно получить доступ к данным материала и объекта. Поэтому вместо общих переменных, используемых в шейдерных языках OpenGL, OptiX позволяет описать любой объект и узел, описанный выше, задав произвольный набор переменных, выраженных как пара ключ-значение.

4. Рассмотрим пример glass из пакета разработчика Optix.

**(слайд код)**

Пример представляет собой класс GlassScene с рядом сопроводительных CUDA процедур и обработчиков запуска.

Обработчиком запуска является процедура main:

int main(int argc, char\* argv[])

{

GLUTDisplay::init( argc, argv );

try {

GlassScene scene( obj\_path, adaptive\_aa, green\_glass );

GLUTDisplay::setTextColor( make\_float3( 0.2f ) );

GLUTDisplay::setTextShadowColor( make\_float3( 0.9f ) );

GLUTDisplay::run( "GlassScene", &scene, adaptive\_aa ? GLUTDisplay::CDProgressive : GLUTDisplay::CDNone );

}

return 0;

}

Как видно из исходного кода, обработчик запуска создает объект класса GlassScene инициализирует его параметрами командной строки и передает его процессору запуска GLUTDisplay. Для корректной работы с GLUTDisplay класс GlassScene должен иметь следующие методы:

**(слайд код)**

void initScene( InitialCameraData& camera\_data );

void trace( const RayGenCameraData& camera\_data );

void doResize( unsigned int width, unsigned int depth );

Buffer getOutputBuffer();

bool keyPressed(unsigned char key, int x, int y);

InitScene создает объект сцены внутри контекста трассировки.

trace используется для построения изображения методом трассировки лучей.

doResize производит необходимые изменения контекста для изменения получаемого изображения.

getOutputBuffer создается выходной буфер, содержащий выходное изображение.

keyPressed--- обработчик клавиш.

**(слайд рис.)**

На основе демонстрационного приложения было создано приложение с помощью метода трассировки лучей.

**(слайд.рис)**

Вывод:

В ходе данной работы изучено следующее:

Программно-аппаратная архитектура CUDA. Архитектура CUDA является довольно простой, но в тоже время мощной технологией для массивно-параллельных вычислений. С помощью архитектуры CUDA практически в режиме реального времени решать задачи по поиску пути в графе, аппроксимировать функции, обрабатывать большие массивы данных и т.д.

Рассмотренны принципы функционирования графического движка OptiX. Он включает реализацию очень широкого набора основанных на трассировке лучей алгоритмов и приложений, включая интерактивный рендеринг, оффлайн рендеринг, системы обнаружения коллизий, запросы искусственного интеллекта и научного моделирования, такие как звуковое распространение.

Механизм OptiX фокусируется исключительно на фундаментальных вычислениях, требуемых для трассировки лучей.

Механизм OptiX демонстрирует, что большинство алгоритмов трассировки лучей могут быть реализованы, используя маленький набор легких программируемых операций. Механизм OptiX комбинирует своевременные методы компиляции со специфичным для трассировки лучей знанием, чтобы реализовать его модель программирования эффективно.

Рассмотрены примеры, входящие в набор разработчика OptiX.

На основе полученной информации разработано демонстрационное приложение.

Таким образом получены достаточная информация и практическое подтверждение того, что Optix является простым, но в тоже время быстрым движком для трассировки лучей общего назначения. Optix можно использовать не только для фотореалистичного рендеринга 3D изображения, но и для задач моделирования корпускулярных и волновых процессов. Это значит, что Optix можно использовать в процессах моделирования звукового распространения, оптических эффектов и так же моделирования ядерных процессов.