Реализация параллельного вычисления серии положений пространственного рычажного механизма на графическом процессоре с помощью DirectX 11 Compute Shader

Е.С. Городецкий

Нижегородский государственный университет им. Н.И. Лобачевского

Введение

Пространственные рычажные механизмы являются важной составляющей современной техники и производственных технологий [4]. Механические системы разрабатываются для выполнения достаточно серьёзных задач, требующие высокой надёжности и точности. Поэтому при конструировании механизма часто требуется выполнять полный расчет его движения для дальнейшей оптимизации модели.

Для автоматизации проектирования пространственных механизмов автором была разработана система “Mechanics Studio” построенная на платформе Microsoft .NET с использованием графической библиотеки Managed DirectX [3]. В программной системе также были реализованы методы последовательного расчета положений механизма.

Наибольший интерес для конструктора представляют технологии и инструменты для оптимального моделирования механических систем. Задача параметрической оптимизации модели рычажного механизма была формализована и описана автором [2]. Дальнейшая работа ведётся в направлении автоматизации решения этой задачи.

Постановка задачи

Для эффективного решения задачи оптимизации механизма требуется применение максимально быстрых алгоритмов расчета положений, реализованных под современные многоядерные архитектуры. Схема параллельных вычислений и структур данных для расчета положений пространственного механизма, была разработана автором в общем виде для CPU и GPU [1]. В данной статье рассматриваются современные технологии вычислений общего назначения на графических процессорах, и выбирается одна из них для реализации разработанного алгоритма. Затем описываются структуры данных и реализация алгоритма для вычисления положений на графической карте.

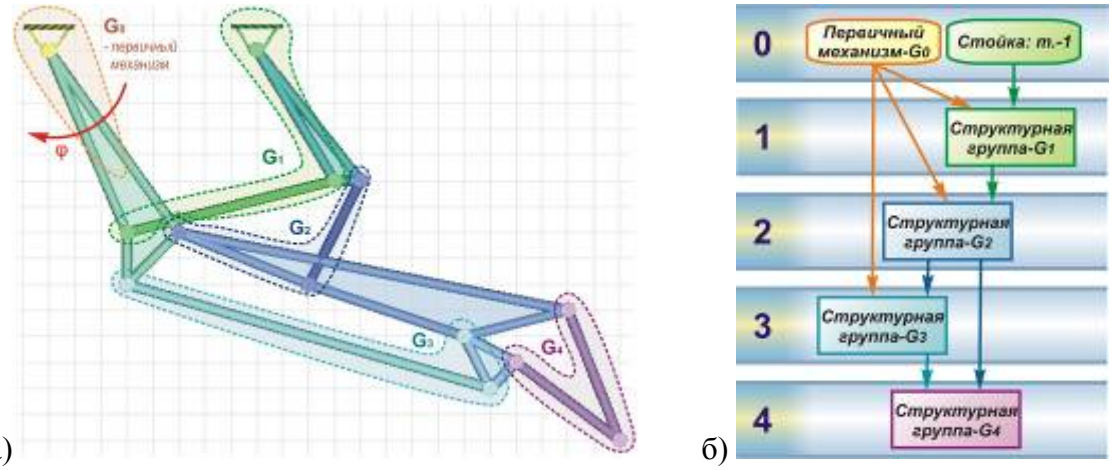
Введение в задачу о положениях пространственного рычажного механизма

Под *механизмом* понимается совокупность взаимосвязанных твёрдых тел, предназначенная для преобразования движения входов на одном или нескольких твёрдых телах в движение на других твёрдых телах (выходах). Механизм может быть представлен своей структурной схемой, на которой выделяют *звенья* (твёрдые тела), соединяющиеся в подвижные *кинематические пары* с помощью различных типов *геометрических элементов* (вращательных, сферических, поступательных и др.). В основе методов конструирования механизмов и расчета кинематики лежит *принцип образования механизмов по Ассуру*, утверждающий что “Всякий механизм представляет собою совокупность одного или нескольких, *двухзвенных* (*первичных*) *механизмов* и одной или нескольких *групп нулевой подвижности* (*структурных групп*)” [4]. В работе рассматривается класс пространственных рычажных механизмов, которые могут быть образованы согласно принципу Ассура наслоением одноконтурных структурных групп.

Прямая задача о положении для механизмов ставится следующим образом. Задано движение всех входных звеньев, удовлетворяющее связям, которые накладывают кинематические пары, соответствующих первичных механизмов. Необходимо найти движение всех остальных звеньев механизма, либо сообщить о невозможности реализации положения, в тот или иной момент времени.

Согласно закону Ассура прямая задача о положениях механизма сводится к последовательному решению ряда более простых подзадач расчета положений структурных

групп механизма. На рис. 1а приведён пример механизма выдвижения закрылков самолёта. Механизм состоит из одного первичного механизма и 4 структурных групп BBB (из 3 Вращательных пар), связанных друг с другом. Каждая группа имеет два входа, необходимых для расчета её положения. Исходя из структурной схемы механизма, строится последовательность расчета структурных групп, показанная на рис. 1б - вычисления производятся последовательно по уровням 0, 1, 2, и т.д., сверху вниз (стрелками показаны соединения групп друг с другом). Положения структурных групп BBB вычисляются аналитически. Произведя вычисления положений механизма для разных углов φ поворота входного звена, мы получим движение всех звеньев механизма.



а) б)

Рисунок 1. Последовательность расчета структурных групп механизма

Важно отметить, что аналитический или численный расчет положения структурной группы механизма требует применения базовых векторных и матричных операций, а также широкого набора геометрических функций на их основе.

Моделирование структурной схемы механизма, разбиение механизма на группы и определение последовательности вычисления структурных групп реализовано в программной системе “Mechanics Studio .NET” [3]. Параллельный расчёт серии положений механизма реализованный на графическом процессоре, выполняется на основе данных полученных от системы моделирования.

Обзор технологий для вычислений общего назначения на графическом процессоре

Вычисления общего назначения на графическом процессоре возникли за счёт возможности применения программируемого графического конвейера для решения вычислительных задач, не связанных с визуализацией. Возможности пиксельных шейдеров (*Pixel Shader*) применялись для обработки данных, упакованных в текстуру, отображаемую на четырёхугольнике в рамках графического конвейера. Для программирования применяются языки *HLSL* и *GLSL*. Несмотря на широкое распространение такого подхода в играх, существует множество ограничений для его применения. Во-первых, требуется создание окна визуализации графики, даже если приложению не нужно ничего рисовать. Во-вторых, данные обрабатываемые пиксельным шейдером должны быть упакованы в текстуру, представляющую собой неструктурированный буфер из 4х-компанентных векторов с плавающей запятой, что создаёт дополнительные сложности по преобразованию данных. И наконец, ввиду архитектурных особенностей графического конвейера, программа пиксельного шейдера может выводить от 1 до 8 4х-компоненентных векторов, записываемых в строго определённые ячейки одной из 8

текстур. Таким образом, главное ограничение подхода пиксельных шейдеров для вычислений общего назначения заключается в строго последовательной записи данных.

Позднее компанией NVidia была разработана специализированная технология *CUDA* для вычислений общего назначения на графических процессорах. Технология *CUDA* реализует параллельную модель *SIMD* (Single Instruction Multiple Data) и позволяет разработчику писать программу обработки данных на C-подобном языке. В отличии от пиксельных шейдеров, данные могут быть структурированы и поддерживается непоследовательная запись данных в массивы. Благодаря своей простоте и гибкости, данная технология получила широкое применение для решения исследовательских задач в академической среде. К сожалению, эта технология поддерживается только на графических процессорах NVidia со специальной версией драйверов. Ещё один недостаток CUDA заключается в отсутствии встроенных векторно-матричных операций и геометрических функций.

За последний год появились сразу две технологии для вычислений общего назначения на графических картах: OpenCL и DirectX Compute Shader. Обе во многом схожи по программной модели с CUDA. OpenCL разрабатывалась как открытый стандарт, который будет поддерживаться обоими крупнейшими производителями графических карт, но пока реализована только в бета-версии ATI Stream SDK.

Microsoft разработала свою технологию вычислительных шейдеров (*compute shader*), в рамках DirectX 11 и HLSL 5.0 [7]. DirectX 11 встроен в новую версию операционной системы Microsoft Windows 7 и в доступен для Windows Vista SP2 в рамках Platform Update. Важно отметить что, несмотря на то, что DirectX 11 разрабатывался для графических карт нового поколения, в нём реализована возможность исполнения приложений с ограниченным набором возможностей (*features sets*) на графических картах более низкого уровня (например, на картах с поддержкой DirectX 10 и 10.1). DirectX 11 поддерживает графические карты как NVidia так и ATI. Вычислительные шейдеры дополняют язык HLSL вычислениями общего назначения SIMD, позволяя выполнять их как в рамках полного графического конвейера, так и вне него (без необходимости выполнять рендеринг).

Для реализации алгоритма параллельного вычисления положений механизма была выбрана технология вычислительных шейдеров, в связи с возможностью использования графических карт различных производителей и применения языка HLSL, уже ставшего стандартом для программирования GPU. Дополнительное преимущество от использования языка HLSL для решения поставленной задачи состоит в наличии встроенных векторных операций и широкого набора геометрических функций (Intrinsic functions), реализованных в железе.

Особенности программной модели Compute Shader

Для работы с вычислительными шейдерами (*compute shader*) в DirectX 11 необходимо создать объект устройства, представляющий графическую карту вызовом функции *D3D11CreateDevice*. При создании устройства есть возможность установки уровня возможностей (*feature level*), поддерживаемого устройством, выбирая из D3D 10.0 / 10.1 / 11.0 в зависимости от поколения графической карты (более низкие поколения не поддерживают вычислительные шейдеры). После создания устройства уровня 10.X следует проверить возможность выполнения вычислительных шейдеров вызвав функцию *CheckFeatureSupport* для проверки доступных функций DirectX 11 (`D3D11_FEATURE_D3D10_X_HARDWARE_OPTIONS`) и убедиться в том что следующий флаг имеет значение True:

`ComputeShaders_Plus_RawAndStructuredBuffers_Via_Shader_4_x`

Далее необходимо выделить память на графической карте под различные типы буферов для входных и выходных данных и создать объекты доступа к буферам:

1. Константный буфер (*constant buffer*), обычно использующийся для передачи шейдеру размеров буферов с данными и скалярных параметров;
2. Структурированные буферы для чтения (*structured buffer*), с массивами входных данных и соответствующими объектами доступа к ресурсам SRV (s*hader resource view*);
3. От 1 до 8 структурированных буферов для записи (RW structured buffer) с соответствующими объектами непоследовательного доступа UAV (*unordered access view*). На устройствах с уровнем возможностей ниже D3D 11 количество таких буферов для записи ограничено одним.

После компиляции HLSL кода (*D3DCompile*), создаётся объект вычислительного шейдера (*CreateComputeShader*) и устанавливается в контексте устройства вместе с всеми вычислительными ресурсами. Наконец, осталось запустить шейдер методом устройства *Dispatch*, указав нужное количество потоков nX*nY*nZ ≤ 1024. Потоки шейдера выполняются группами в соответствии с кодом шейдера. Каждая группа имеет разделяемую кэш-память: на устройствах D3D 10 до 16 KB, на D3D 11 до 32 KB [6].

Структуры данных и вычислительная схема

На рис. 2 показаны структуры данных используемые для вычисления положений механизма с помощью вычислительного шейдера:

- CS_MECHANISM – содержит размеры буферов и массивов;
- CS_ITERATION – одно положение механизма с массивами точек и осей;
- CS_LINK – звено механизма с массивами индексов его точек и осей;
- CS_GROUP – содержит данные о структурной группе механизма.

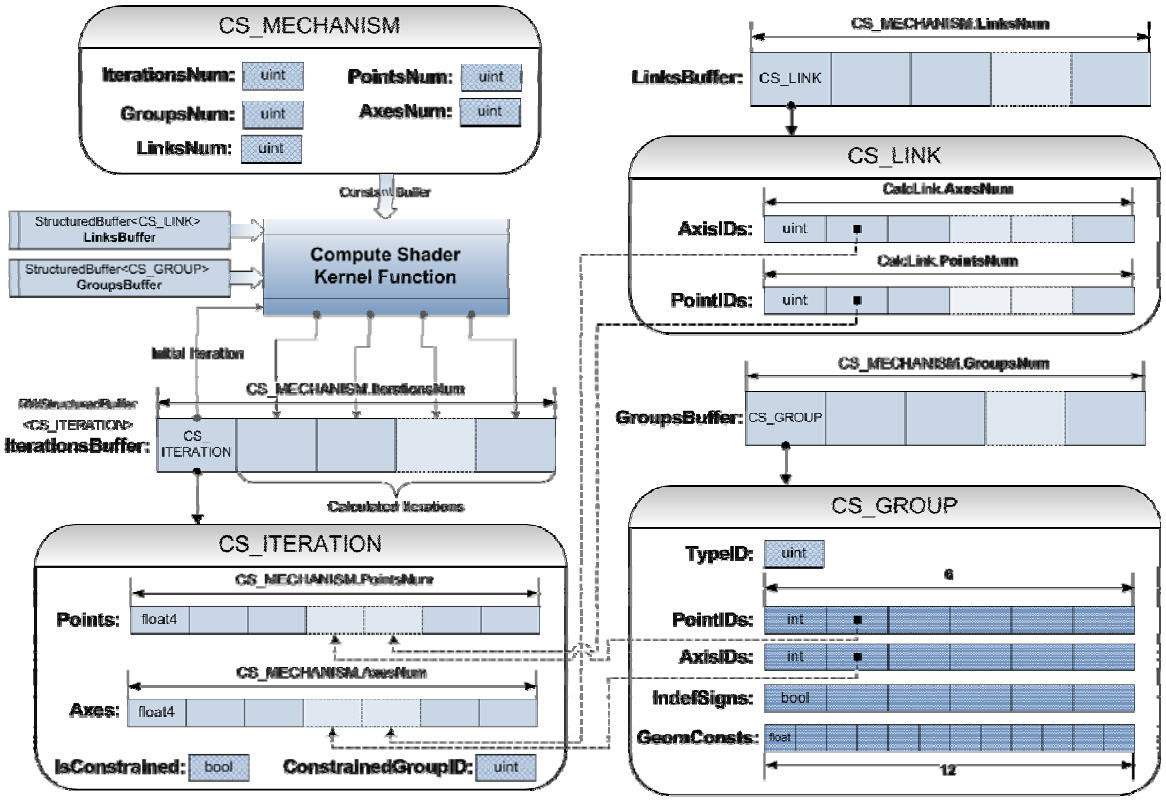


Рисунок 2. Структуры данных для параллельного вычисления положений механизма

Вычисление положений выполняется функцией ядра вычислительного шейдера, принимающей на вход константный буфер типа CS_MECHANISM, и структурированные буферы для чтения: *LinksBuffer* и *GroupsBuffer*. До вычислений, при инициализа-

ции выходного структурированного буфера *IterationsBuffer* в 0й элемент записывается начальное положение механизма. Каждый поток вычисляет одно из положений {1,…, IterationsNum-1} на основе входных данных о звеньях, группах и начального положения и записывает вычисленные массивы точек и осей положения в элемент выходного структурированного буфера *IterationsBuffer*. Более подробно алгоритм вычисления положений описан в [1].

Заключение

Разработанные структуры данных и схема вычислений описывают параллельный алгоритм расчёта серии положений механизма, в котором каждое положение вычисляется независимо. Алгоритм реализован с использованием технологии вычислительных шейдеров для выполнения на современных графических процессорах с сотнями векторных вычислительных ядер, чтобы добиться значительного ускорения при вычислениях серии положений механизма. Данные по производительности данной реализации будут приведены в материалах выступления.

Высокопроизводительное вычисление положений механизма разработано для дальнейшего применения в решении задачи параметрической оптимизации рычажных механизмов, где для вычисления одного значения целевой функции требуется вычислить серию из десятков или сотен положений конфигурации механизма [2].

Работа выполнена при финансовой поддержке Федерального агентства по науке и инновациям (ФЦП «Научные и научно-педагогические кадры инновационной России», госконтракт № 02.740.11.5018).

Литература

1. Городецкий Е.С. Алгоритм параллельного вычисления серии положений пространственного рычажного механизма // Технологии Microsoft в теории и практике программирования: Материалы конференции. Н. Новгород: Изд-во Нижегородского госуниверситета, 2009г. - стр. 102-108;
2. Городецкий Е.С. Задача параметрической оптимизации пространственных механизмов с параллельными вычислениями кинематики на графической карте // Высокопроизводительные параллельные вычисления на кластерных системах: Материалы седьмой международной конференции-семинара. Н.Новгород: Изд-во Нижегородского госуниверситета, 2007г. – стр. 117-124;
3. Городецкий Е.С. Система проектирования кинематики пространственных механизмов Mechanics Studio .NET на основе Managed DirectX // Технологии Microsoft в теории и практике программирования: Материалы конференции. Н. Новгород: Изд-во Нижегородского госуниверситета, 2007г., – стр. 48-52;
4. Заблонский К.И., Белоконев И.М., Щекин Б.М. “Теория механизмов и машин: учеб. для вузов” - К.: Выща шк. Головное изд-во, 1989.- 376с.
5. Турлапов В.Е. “Задача о положениях и классификация одноконтурных структурных групп пространственных рычажных механизмов” // Электронный ж. "Прикладная геометрия". Вып.2. №2. МАИ. 2000г. С.1-22;
6. Nick Thibieroz (AMD) Shader Model 5.0 and Compute Shader: GDC Conference materials, 2009 [www.gdconf.com/conference/Tutorial Handouts/100_Advanced Visual Effects with Direct3D/100_Handout 6.pdf]
7. Microsoft DirectX Software Development Kit (August 2009) [msdn.microsoft.com/en-us/directx/default.aspx]