Программирование современных графических процессоров с использованием основных языков платформы .NET

Березин Сергей, Калугин Константин, Карпушина Екатерина, Невский Евгений, Носов Кирилл, Павлова Ольга
Department of Computational Mathematics and Cybernetics
Moscow State University, Moscow, Russia

Аннотация

Шейдер – это процедура, управляющая преобразованием координат и атрибутов вершин (вершинные шейдеры) и закрашиванием объектов на уровне отдельных пикселей (пиксельные шейдеры). Мы представляем проект, находящийся на стадии реализации, по разработке транслятора из промежуточного языка Microsoft IL в низкоуровневый язык шейдеров. Работа над проектом ведется при финансовой поддержке Microsoft Research.

Keywords: *Вершинные и пиксельные шейдеры, Графические процессоры, Microsoft Intermediate Language, Регистровые и стековые машины, JIT- компиляторы.*

1. ВВЕДЕНИЕ

Большинство современных интерактивных графических приложений основано на использовании специализированных графических процессоров (GPU) для обработки полигональной графики. Общая схема работы GPU может быть представлена при помощи модели графического конвейера (рис. 1).

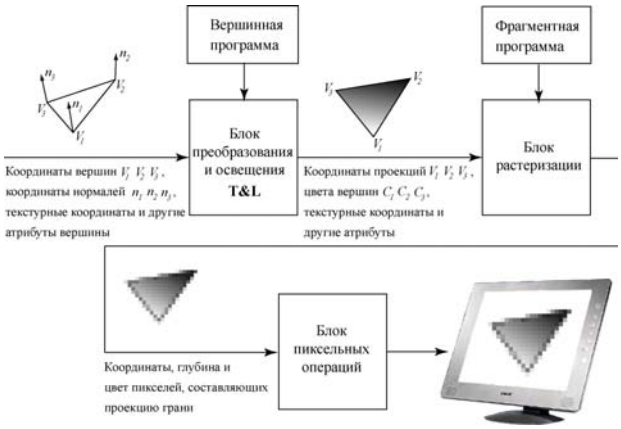

Рисунок 1: Программируемый графический конвейер.

Почти все современные графические процессоры являются программируемыми и поддерживают два типа шейдеров: вершинные и фрагментные (пиксельные). Применение шейдеров позволяет в реальном времени создавать сложные визуальные эффекты, включающие нетривиальный расчет освещения для каждого пикселя, имитацию неровных поверхностей при помощи карт нормалей, моделирование отражающих и преломляющих объектов, расчет динамических теней и объемного тумана и многие другие эффекты, доступные ранее только для алгоритмов трассировки лучей (рис. 2).

Рисунок 2: Визуальные эффекты с применением шейдеров.

Языки для программирования шейдеров можно разделить на низкоуровневые и высокоуровневые.
К низкоуровневым языкам можно отнести языки вершинных и пиксельных шейдеров Direct3D [2] и языки, определяемые расширениями OpenGL ARB_vertex_program и ARB_fragment_program. Для каждого языка определена своя архитектура абстрактной регистровой машины. Все архитектуры обладают рядом общих свойств:

- Большинство регистров являются векторными и хранят четыре действительных числа.
- Большинство команд предназначено для выполнения векторых операций, хотя возможно изменение отдельных компонент векторов.
- Стековой памяти не предусмотрено. Косвенная адресация реализована исключительно в виде выборки векторного или скалярного значения из текстуры.
- Нет возможности вызова подпрограмм. Возможности ветвления ограничены.
- Регистры разделены на входные, выходные, временные и константные. Константные регистры остаются неизменными при обработке всего блока

геометрических данных (трехмерного объекта) и используются для настройки режимов работы шейдера.

Компиляцию и загрузку шейдерной программы на низкоуровневом языке выполняют графическая библиотека и драйвер видеокарты.

При использовании низкоуровневых языков возникают следующие неудобства, часть из которых характерна для всех языков ассемблера:

- Трудоемкое кодирование даже простых алгоритмов.
- Зависимость программы от архитектуры (в данном случае от графической библиотеки иди процессора). Большие затраты при переписывании программы на другом языке.
- Необходимость написания специального кода для интеграции пиксельных и вершинных программ в основную.

К высокоуровневым языкам можно отнести Microsoft High Level Shader Language [3], NVidia CG [4-6] и OpenGL 2.0 Shader Languages. Все высокоуровневые языки являются процедурными и обычно транслируются в один из низкоуровневых языков. Часть ограничений, связанных с низкоуровневостью языков снимается, но прикладная программа по прежднему должна загрузить шейдерную программу из файла, задать значения используемых в шейдерной программе констант и, возможно, связать шейдерную программу с графическим процессором.

Для устранения указанных выше недостатков предлагается разработать транслятор из промежуточного языка Microsoft IL в низкоуровневый язык шейдеров (в качестве целевого языка рассматривается язык шейдеров Direct3D).

2. ХАРАКТЕРИСТКА ИСХОДНОЙ И ЦЕЛЕВОЙ ПЛАТФОРМ

Библиотека Direct3D поддерживает несколько версий языков шейдеров. Основные различия заключаются в поддержке инструкций ветвления, ограничениях на максимальную длину программы и количество регистров. Характеристики этих версий сведены в таблице 1.

Таблица 1: . *Версии вершинных шейдеров Direct3D.*

ps_1_1 – ps_1_4	Максимум 128 инструкций; нет инструкций ветвления; 12 временных регистров; 96 константных регистров
vs_2_0	Максимум 256 инструкций; статические инструкции ветвления; 12 временных регистров; 256 константных регистров
vs_2_x, vs_3_0	Не менее 512 инструкций; динамические инструкции ветвления; 12 временных регистров; 256 константных регистров
ps_1_1 – ps_1_4	Максимум 14 инструкций; нет инструкций ветвления; 6 временных регистров; 8 константных регистров
ps_2_0	Максимум 96 инструкций; статические инструкции ветвления; 12 временных регистров; 32 константных регистров
ps_3_0	Не менее 512 инструкций; динамические инструкции ветвления; 32 временных регистров; 256 константных регистров

Исходный язык Microsoft IL [7] является языком абстрактной стековой машины, содержащим достаточно высокоуровневые инструкции для создания объектов, вызовов методов объектов и обработки исключительных ситуаций. Для вычислений используется стек, предусмотрены специальные инструкции для работы с локальными переменными.

Возможности языка Microsoft IL значительно шире возможностей шейдерных языков. Чтобы трансляция стала возможной, набор поддерживаемых инструкций ограничивается следующим образом:

- Не поддерживаются инструкции обработки исключений.
- Ограничены возможности инструкции вызова (нет виртуальных вызовов, поддерживаются либо вызовы методов из заранее определенного подмножества, либо вызовы, которые могут быть развернуты при помощи макроподстановки).
- Набор используемых типов ограничен встроенными целочисленными и вещественными типами, а также структурами для представления векторов и матриц размерностью 3 и 4.

Перечисленные выше ограничения с одной стороны делают задачу трансляции MSIL в языки шейдеров разрешимой на практике, с другой стороны не слишком сильно ограничивают возможности программиста по разработке щейдерных программ.

3. МЕТОДЫ РЕШЕНИЯ

Задачу трансляции MSIL в язык шейдеров предлагается решать в два этапа. Сначала производится перевод кода MSIL в промежуточное представление, что может быть сделано один раз при первом обращении к шейдеру. Затем промежуточное представление транслируется в требуемую версию шейдерного языка. На этапе трансляции в промежуточное представление выполняются следующие действия:

- Проверяется наличие неподдерживаемых инструкций MSIL.
- Вызовы методов заменяются на векторные и математические операции или разворачиваются при помощи макроподстановки.
- Особым образом помечаются входные регистры, выходные регистры и константные параметры.

В качестве промежуточного представления предлагается использовать код абстрактной трехадресной машины с неограниченным количеством временных регистров. Методы для трансляции стековой архитектуры в регистровую могут быть разработаны на основе применяемых в существующих

JIT компиляторах [8-10]. Отсутствие стековой памяти компенсируется неограниченным количеством временных регистров. Основной задачей при трансляции промежуточного представления в целевой язык является задача распределения ограниченного количества регистров. Общую постановку задачи распределения регистров и некоторые методы решения можно найти в [11-14].

Помимо основной задачи перевода MSIL в языки шейдеров решаются некоторые вспомогательные задачи технического характера. В частности, атрибуты .NET применяются для назначения полей вершин или привязки полей вершин к входным и выходными параметрам шейдеров.

4. ПРИМЕР ШЕЙДЕРНОЙ ПРОГРАММЫ

Рассмотрим в качестве примера реализацию смешения матриц (matrix blending). Основная идея такова: каждой вершине объекта присваивается вес $\alpha_i \in [0,1]$. Задаются две матрицы преобразования M_1 и M_2. Преобразованные координаты вершины v'_i вычисляются по формуле

$$v'_i = \alpha_i M_1 v_i + (1-\alpha_i) M_2 v_i .$$

Таким образом, на часть точек объекта большее влияние оказывает матрица M_1, на другие точки – матрица M_2. Плавное изменение веса вдоль объекта позволяет выполнять достаточно сложную анимацию и изменение формы объекта.

На рис. 4 показаны результаты смешения матриц, где в качестве объекта взят куб, веса вершин которого линейно увеличиваются от 0 до 1 вдоль оси OY (веса точке отмечены на рис. 4 цветом), а в качестве матриц M_1 и M_2 взяты матрицы поворота вокруг оси OY на углы β и $-\beta$.

Необходимо отметить, что искажение формы объекта происходит без изменения исходного геометрического описания объекта, что позволяет значительно сократить объем передаваемых GPU данных и увеличить производительность программы.

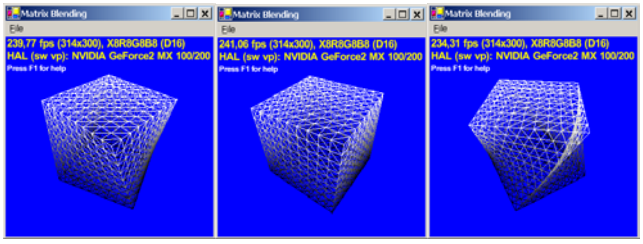

Рисунок 4: *Изменение геометрии при помощи смешения матрицы*

Примерный вид исходного текста шейдера на языке C# показан на листинге 1.

```
[StructLayout(LayoutKind.Sequential)]
public struct WeightedVertex {
    [Position(0)]
    public float3 Position;
    [BlendWeight(0)]
    public float Weight;
}

public struct V2F {
    [Position]
    public float4 Position;
    [Color]
    public float4 Color;
}

class BlendShader : VertexShader {
    public float4x4 Trasnform1;
    public float4x4 Transform2;

    public V2F Main(WeightedVertex v) {
        V2F result = new V2F();
        result.Color = new float4(v.Weight,
                      v.Weight,v.Weight,v.Weight);
        result.Position =
            Transform1*v.Position*v.Weight +
            Transform2*v.Position*(1 - v.Weight);
        return result;
    }
}
```

Листинг 1. Шейдер на языке C#.

5. ВЫВОДЫ

Применение такого транслятора делает возможным использование многих языков программирования, компиляторов и другого инструментария разработчика на платформе .NET для программирования современных GPU;

написание шейдеров на языке C# упростит исходный код в частности и общее управление проектом в целом. Кроме того

транслятор MSIL в языки шейдеров может оказаться полезным при создании высокоуровневой объектно-ориентированной модели графического конвейера, которая может быть запрограммирована целиком на языке C# или другом языке платформы .NET.

Работающий прототип транслятора будет закончен к сентябрю 2004 года.

Адрес сайта проекта: microsoft.cs.msu.su/projects/ilshaders

6. ССЫЛКИ

[1]www.gpgpu.org

[2]DirectX programming pipeline: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/programmingguide/Programmable/PixelShaders/PixelShaders.asp

[3]High Level Shader Language: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/directx9_c/directx/graphics/reference/HighLevelLanguageShaders.asp

[4]NVidia CG language home page: http://www.cgshaders.org/

[5]William R. Mark, R. Steven Glanville, Kurt Akeley, Mark J. Kilgard. CG: A system for programming graphics hardware in C-like language.

[6]www.cgshaders.org

[7]Serge Lidin. Inside Microsoft .NET IL Assembler, Microsoft Press 2002

[8]Byung-Sun Yang et al, "LaTTe: A Java VM Just-in-Time Compiler with Fast and Efficient Register Allocation," Proceedings of the 1999.

[9]Ali-Reza Adl-Tabatabai, Michal Cierniak, Guei-Yuan Lueh, Vishesh M. Parikh, and James M. Stichnoth. Fast, effective code generation in a just-in-time Java compiler. In PLDI '98: Proceedings of the ACM SIGPLAN '98 Conference on Programming Language Design and Implementation, pages 280-290, Montreal, 1998. ACM.

[10]Andreas Krall. Efficient JavaVM just-in-time compilation. In Proceedings of the 1998 International Conference on Parallel Architectures and Compilation Techniques, Paris, France, October 1998.

[11]А. Ахо, Д. Ульман, Р. Сети. Компиляторы: принципы, технологии и инструментарий, Вильямс 2001.

[12]В. А. Серебряков. Лекции по конструированию компиляторов: http://www.ergeal.ru/archive/cs/cc.htm.

[13]G. J. Chaitin. Register allocation & spilling via graph coloring. Proceedings of the 1982 SIGPLAN symposium on Compiler construction.

[14]M. Poletto. Linear scan register allocation. ACM Transcations on Programming Languages and Systems, Vol. 21 No. 5.

Об авторах.

Сергей Березин работает ассистентом на факультете Вычислительной математики и кибернетики Московского государственного университета им. М.В. Ломоносова. Адрес электронной почты: s_berezin@cs.msu.su.

Екатерина Карпушина является выпускником факультете Вычислительной математики и кибернетики Московского государственного университета им. М.В. Ломоносова. Адрес электронной почты: csnake@mail.ru.

Константин Калугин – студент факультета Вычислительной математики и кибернетики Московского государственного университета им. М.В. Ломоносова. Адрес электронной почты: kg@cs.msu.su.

Кирилл Носов - студент факультета Вычислительной математики и кибернетики Московского государственного университета им. М.В. Ломоносова. Адрес электронной почты: kirill_nosov@mail.ru.

Невский Евгений – студент факультета Вычислительной математики и кибернетики Московского государственного университета им. М.В. Ломоносова.