Оптимизация определения видимых фрагментов текстуры для алгоритма виртуализации памяти графических ускорителей

Гордеев Антон, Денис Гладкий, Игорь Белаго, Станислав Кузиковский
Институт Автоматики и Электрометрии СО РАН
Новосибирск, Россия
togorde@gmail.com, generalgda@gmail.com, {bel, stas}@sl.iae.nsk.su

Аннотация

Данная статья посвящена алгоритму виртуализации памяти графических ускорителей. Он используется при реализации фотореалистичного текстурирования. Для этого виртуальный мир текстурируется одной большой текстурой, разрешение которой может превосходить 128000 на 128000 пикселей.

Для определения видимых фрагментов текстуры, которые необходимо загружать в видеопамять для формирования кадра сцены, используется идея построения текстурной карты – таблицы индексов вышеупомянутых фрагментов.

В данной статье предложен метод оптимизации конструирования текстурной карты. Для этого используются возможности архитектуры Shader Model 4.0, а именно геометрический шейдер (geometry shader).

Ключевые слова: *Виртуализация памяти, Графический ускоритель, Фотореалистичное текстурирование, Megatexture, Virtual texture mapping.*

1. ВВЕДЕНИЕ

Современные компьютерные игры и другие интерактивные приложения виртуальной реальности, для достижения фотореалистичности изображения, используют большие объемы высоко детализированных текстурных данных. Поэтому на видеокарты накладываются постоянно увеличивающиеся серьёзные требования на объём видеопамяти. Также эти требования постоянно увеличиваются. Однако объём доступной графической памяти не растет в таком же темпе. Более того, память графических ускорителей требуется для хранения вершинных и индексных буферов, осуществления сложных эффектов пост-обработки, расчёта освещенности сцены и многих других задач.

2. ВИРТУЛАЗИЦАЯ ПАМЯТИ ГРАФИЧЕСКИХ УСКОРИТЕЛЕЙ

Алгоритм виртуализации памяти графических ускорителей предназначен для уменьшения объёма памяти видеокарты, необходимой для размещения текстур, до предела, который в идеальном случае определяется разрешением экрана (для закраски N пикселей уникальными цветами необходимо, как минимум, O(N) байтов памяти – по блоку фиксированного размера на каждый пиксель).

Для текстурирования мира используется одна большая текстура, разрешение которой может достигать 128000 на 128000 пикселей [1]. Её несжатое представление может занимать объём, превышающий несколько гигабайт. Такая текстура не может целиком поместиться в видеопамять современных графических ускорителей.

Основная идея алгоритма заключается в том, что для каждого конкретного кадра не требуется загружать всю текстуру целиком, необходимы только те её фрагменты, которые будут видны в данном кадре на экране.

3. ПОСТРОЕНИЕ ТЕКСТУРНОЙ КАРТЫ

Для определения видимых фрагментов текстуры используется алгоритм построения текстурной карты в отдельном предварительном цикле работы видеокарты. При этом используется идея рендеринга геометрии в текстурных координатах [4]. Эта идея заключается в том, что в качестве геометрических координат вершины используются её текстурные координаты, а растеризация ведётся в специальный буфер. После завершения работы конвейера видеокарты, каждый фрагмент геометрии растеризуется в области, использующейся для его (фрагмента) текстурирования.

Поскольку, у каждой вершины вместо реальных геометрических координат используются её текстурные координаты, то в процессе работы графического конвейера невозможно использовать ряд его автоматических тестов отбраковки растеризуемых фрагментов, например:

- frustum culling
- back face culling
- z test

Поэтому необходимо эмулировать их с помощью программируемых блоков конвейера видеокарты.

В результате получается текстурная карта, закрашенные пиксели которой соответствуют участкам исходной большой текстуры, которые необходимо загрузить в видеопамять.

4. ОПТИМИЗАЦИЯ ПОСТРЕНИЯ ТЕКСТУРНОЙ КАРТЫ

В ситуации, когда объекты перекрывают друг друга относительно камеры (occlusion culling) выделяются два возможных случая. В первом, фрагмент одного объекта перекрывается фрагментом другого. Во втором, фрагмент попадает на заднюю грань объекта, и перекрывается его передней гранью. Отбраковка фрагмента, при его попадании на заднюю грань, называется back face culling.

Для выполнения back face culling нами был разработан новый подход, базирующийся на геометрическом шейдере – этапе графического конвейера, доступного для видеокарт, реализующих архитектуру Shader Model 4.0 [3]. Он позволяет написать программу, обрабатывающую целый геометрический примитив (отрезок, треугольник и пр.), а не отдельные пиксели или вершины, как позволяли графические ускорители архитектуры Shader Model 3.0 [2]. Подход также может быть полезен при использовании различных алгоритмов, нестандартно использующих back face culling совместно с методом виртуального текстурирования.

Идея оптимизации заключается в том, что в геометрическом шейдере можно реализовать проверку, попадает ли треугольник на переднюю или заднюю грань объекта. По результату проверки, треугольник либо сбрасывается, либо передаётся на следующий этап графического конвейера. Сама проверка, как таковая,

может быть реализована в виде сравнения знака скалярного произведения нормали, обрабатываемого треугольника, с вектором от любой его точки к позиции камеры (вектор «на камеру»). Нормаль к треугольнику может строиться как статически, передаваясь на конвейер видеокарты в виде атрибута вершины, так и вычисляться динамически. В нашей реализации (листинг 1) мы использовали второй подход.

Стоит заметить, что, как вектор «на камеру», так и нормаль треугольника, не обязательно должны быть нормированным, поскольку знак скалярного произведения не зависит от длин векторов-операндов.

```
[maxvertexcount(3)]
void geometryShader(triangle GSPS_INPUT input[3],
   inout TriangleStream<GSPS_INPUT> OutputStream)
{
   float3 edge1 = input[1].ActualPos - input[0].ActualPos;
   float3 edge2 = input[2].ActualPos - input[0].ActualPos;
   float3 normal = cross(edge1, edge2);
   float3 toCamera = eye - input[1].ActualPos;
   if (dot(toCamera, normal) >= 0.0f)
   {
      for(int i = 0; i < 3; i++)
      {
         OutputStream.Append( input[i] );
      }
      OutputStream.RestartStrip();
   }
}
```

Листинг 1: код геометрического шейдера на языке HLSL.

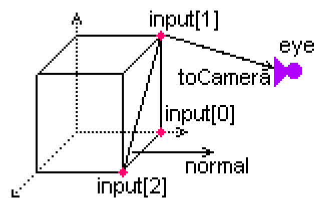

Рисунок 1: пример грани, прошедшей тест в геометрическом шейдере.

Степень ускорения построения текстурной карты, при использовании описанной оптимизации, зависит, как от конфигурации сцены, так и от положения камеры на ней. Можно выделить два краевых случая. В обоих из них сцена представляет собой график функции, заданной на подмножестве R^2. Примером может служить часто встречающиеся в приложениях такие односторонние полигональные сетки, как водная поверхность или ландшафт. В вырожденном случае первого рода (рисунок 2) камера находится над поверхностью и направлена «на неё» (скалярное произведение векторов направления «на камеру» и нормали любого треугольника поверхности больше нуля).

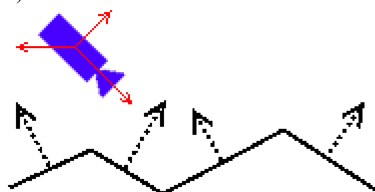

Рисунок 2: вырожденный случай 1-го рода.

В этом случае оптимизация не только не даст улучшения производительности, но и может снизить её, поскольку ни один треугольник сцены не будет отбракован, не смотря на затраченные дополнительные вычисления.

В вырожденном случае второго рода (рисунок 3) камера находится «под поверхностью» (скалярное произведение вектора направления «на камеру» и нормали любого треугольника поверхности меньше нуля), в результате чего будут отбраковываться все треугольники сцены, что полностью разгружает этапы графического конвейера, следующие за геометрическим шейдером (растеризация, пиксельный шейдер и пр.).

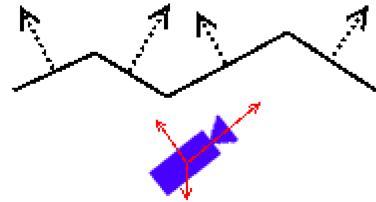

Рисунок 3: вырожденный случай 2-го рода.

Однако типичная сцена современных интерактивных приложений виртуальной реальности редко представляет собой описанные вырожденные случаи. Чаще всего она формируется из выпуклых многогранников или объектов, составленных из них. В подобной сцене, в среднем, стоит ожидать ускорение построения текстурной карты в два раза, благодаря отбраковке, примерно, половины треугольников.

5. РЕЗУЛЬТАТЫ ИЗМЕРЕНИЙ

Для тестирования производительности были произведены измерения количества кадров, рисуемых за секунду, при построении текстурной карты. Были использованы фрагменты виртуального мира из игры "Дальнобойщики 3", состоящие из различного числа треугольников. Измерения производились при фиксированном пролете видеокамеры. Использовалась видеокарта AMD Radeon 6770m.

Результаты измерений показали, что вышеописанная оптимизация, как и ожидалось, позволяет получить выигрыш в производительности близкий к 50%.

В таблице 1 приведены результаты измерения количества кадров, рисуемых за секунду при построении текстурной карты с вышеупомянутой оптимизацией и без неё.

Конфигурация сцены	С гш*	Без гш
Модель 1 (4500 треуг-ов)	935	934
Модель 2 (11000 треуг-ов)	775	483
Модель 3 (20000 треуг-ов)	336	189
Модель 4 (70000 треуг-ов)	225	139

Таблица 1, * - *геометрический шейдер*.

Отсутствие прироста производительности при использовании оптимизации на сцене в первой записи таблицы (конфигурация «Модель 1») объясняется строением сцены. Она представляет собой ландшафт с расположенной на нём полигональной моделью жилого здания. Поскольку количество треугольников в последнем мало по сравнению с их общим числом на сцене, то данная конфигурация является близкой к вырожденному случаю первого рода.

В остальных строках таблицы представлены результаты тестирования на сценах, содержавших ту же ландшафтную сетку, но с уже более сложными (по количеству

треугольников) архитектурными объектами, что является более типичным случаем для современных приложений.

6. ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы был реализован алгоритм определения видимых фрагментов текстуры для виртуализации памяти графических ускорителей. Для него была предложена и реализована оптимизация, отбрасывающая задние грани в геометрическом шейдере. Были проведены измерения производительности, показавшие положительные результаты.

7. ССЫЛКИ

[1] Matthaus G. Chajdas, Christian Eisenacher, Marc Stamminger, Sylvain Lefebvre. Virtual Texture Mapping 101. GPU Pro: advanced rendering techniques / edited by Wolfgang Engel. A K Peters, Ltd. 2010.

[2] Randima Fernando. Shader Model 3.0 Unleashed. SIGGRAPH Proceedings, 2004.

[3] Suryakant Patidar, Shiben Bhattacharjee, Jag Mohan Singh, P. J. Narayanan. Exploiting the Shader Model 4.0 Architecture. IIIT/TR/2007/145, March 2007.

[4] Sylvain Lefebvre, Jerome Darbon, Fabrice Neyret. Unified Texture Management for Arbitrary Meshes. Technical Report RP5210, INRIA, 2004.

Об авторах

Гордеев Антон – студент кафедры АФТИ ФФ НГУ. Его адрес: togorde@gmail.com.
Денис Гладкий – ассистент кафедр ИИС ФИТ и АФТИ ФФ НГУ, инженер-программист компании Playtox. Его адрес: generalgda@gmail.com.
Игорь Белаго – научный сотрудник лаборатории №13 ИАиЭ СО РАН. Его адрес: bel@sl.iae.nsk.su.
Станислав Кузиковский - научный сотрудник лаборатории №13 ИАиЭ СО РАН. Его адрес: stas@sl.iae.nsk.su.