Обработка изображений в браузерных приложениях на потоковых графических процессорах

Андрей Сморкалов
Лаборатория систем мультимедиа
Марийский Государственный Технический Университет, Йошкар-Ола
asmorkalov@mail.ru

Аннотация

В статье рассматриваются подходы к организации обработки изображений в браузерных приложениях с использованием вычислительных возможностей графических потоковых процессоров. Идея применения потоковых процессоров в этом случае имеет большие перспективы, т.к. скорость выполнения javascript-кода значительно уступает скорости выполнения программ на таких языках, как C/C++, а пиковая производительность потоковых процессоров в десятки раз выше производительности центральных процессоров.

Ключевые слова: *Обработка изображений, Браузерные приложения, Потоковые процессоры, WebGL.*

1. ВВЕДЕНИЕ

Графические потоковые процессоры характеризуются большой степенью параллелизма, обладая сотнями и даже тысячами вычислительных ядер, но накладывают на выполняемые алгоритмы большое число ограничений (в основном, из-за изначальной ориентированности на задачи 3D-графики), которые делают невозможным прямой перенос алгоритмов обработки изображений для выполнения на потоковых процессорах. Среди этих ограничений отсутствие рекурсии, прямого доступа к памяти на запись, неопределенный результат чтения из результирующего изображения и др.

Между тем, превосходство потоковых над обычными процессорами в десятки раз по вычислительной мощности делает перспективным обработку двумерных изображений именно на потоковых процессорах [6].

Не менее перспективной является идея использования потоковых процессоров для обработки изображений в браузерных приложениях в связи с их ограниченным доступом к вычислительным возможностям центрального процессора. Прямое исполнение кода на центральном процессоре из браузерного приложения невозможно без использования специально разработанного дополнительного модуля для браузера (плагина). Механизм поддержки плагинов варьируется от браузера к браузеру, что вносит дополнительные трудности и требует разработки плагинов под каждый поддерживаемый браузер.

Актуальным является разработка программной системы для браузерных приложений, предлагающей удобную абстракцию для решения основных задач обработки изображений. Одним из основных достоинств такой системы будет то, что от программиста при решении задач обработки изображений не потребуется знания аппаратных особенностей потоковых процессоров и умения программировать их на низком уровне.

2. ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ

Долгое время обработку изображений в браузерных приложениях можно было реализовать с помощью двух основных подходов: с использованием Java-апплета и с использованием Flash-компонента. Так, например, для обработки изображений в java-апплетах широко используется библиотека ImageJ [1].

При использовании обоих подходов требуется установка специальных плагинов, что является во многих случаях нежелательным или невозможным (по соображениям безопасности, ввиду отсутствия плагина для нужной платформы или браузера, ввиду недостаточной квалификации пользователя для установки, а также по другим соображениям). Кроме того, в обоих случаях обработка изображений производится байт-кодом, выполняемым на виртуальной машине Flash или Java. Таким образом, скорость обработки изображений будет значительно уступать скорости программ на компилируемых языках, таких как C/C++, и тем более скорости обработки изображений на графических потоковых процессорах.

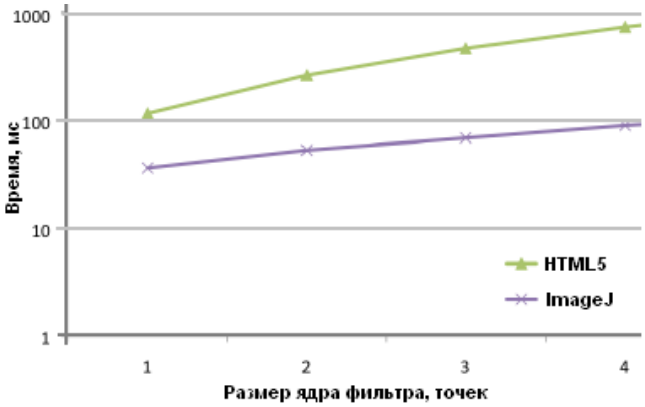

Рис 1. Сравнение производительности box-фильтра с помощью HTML5 canvas и ImageJ.

С появлением стандарта HTML5 стала возможна обработка изображений непосредственно на языке javascript с использованием элемента canvas [3]. Преимущество метода заключается в отсутствии необходимости установки каких-либо плагинов, однако

1. Программа обработки изображений выполняется на виртуальной машине javascript.

2. Использование многоядерности процессоров затруднено.

Эти факторы негативно отражаются на производительности подхода. По приводимому в [3] графику видно, что скорость описываемого подхода в несколько раз уступает скорости обработки изображений с помощью ImageJ (рис. 1).

В настоящее время все большее распространение получает технология PixelBender [2], реализованная в Adobe Flash начиная с версии 10.0. Эта технология позволяет в ограниченном объеме использовать возможности потоковых процессоров, разрабатывая программы для них на специальном языке, специфичном для Flash.

Существенные минусы PixelBender:

1. Технология реализована внутри Flash, а значит требует установки плагина и имеет все перечисленные проблемы технологий, требующих плагинов.

2. Специальный язык ограничивает доступ к вычислительным возможностям потоковых процессоров, ограничивает простор для оптимизации и т.д.

10 февраля 2011 года была опубликована спецификация открытого стандарта WebGL [5] – реализации OpenGL ES 2.0 для использования в браузерных приложениях. WebGL дает возможность разрабатывать OpenGL-программы непосредственно на javascript, включая использование вершинных и пиксельных шейдеров. При этом становится доступной вся функциональность потоковых процессоров.

Для использования WebGL не требуется установки дополнительных модулей, поддержка технологии реализуется непосредственно в браузере. Перечисленные качества делают технологию перспективной для организации обработки изображений в браузерных приложениях.

3. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

Составим математическую модель обработки изображений, опираясь на особенности и возможности WebGL. Будем рассматривать изображение в цветовой модели RGBA:

$U(x, y) = \{f_R(x, y), f_G(x, y), f_B(x, y), f_A(x, y)\}$, (1)

где $f_R(x, y)$, $f_G(x, y)$, $f_B(x, y)$, $f_A(x, y)$ — это дискретные функции, заданные табличным методом, x, y – координаты внутри изображения. $f_R(x, y)$ представляет красный канал изображения, $f_G(x, y)$ - зеленый, $f_B(x, y)$ - синий, $f_A(x, y)$ - альфа-канал. Значения этих функций лежат в диапазоне [0, 1]. $x \in [0, w-1)$, $y \in [0, h-1)$, где w и h – ширина и высота изображения.

Результатом преобразования G изображения A на основе изображения B будем называть R = G(A, B, x, y), (2) где A — это исходное изображение, B – растеризуемая фигура, G — преобразующая функция.

Цветовой маской будем называть множество $M = \{M_R, M_G, M_B, M_A\}$ (3), где M_R, M_G, M_B, M_A могут принимать значения из множества {0, 1}. Результатом преобразования G изображения A на основе изображения B с учетом цветовой маски M будем называть $R_q = G_q(A, B, M, x, y) = G_q(A, B, x, y) * M_q + A * (1 – M_q)$ (4), где q – любой из цветовых каналов.

Геометрическая фигура $S = \{V, F, \{r, g, b, a\}\}$ (5) задается множеством двумерных векторов вершин V и множеством индексов вершин F, а также цветом фигуры {r, g, b, a}. Растеризация представляет собой преобразование, результатом которого является изображение $U(x, y) = G_R(G_P(S, M_P))$ (6), где G_R – растеризующее преобразование, M_P – матрица проецирования 4x4, G_P — проецирующее преобразование.

В результате преобразования G_P получается спроецированная фигура $S_P = G_P(S, M_P) = \{P, F, \{r, g, b, a\}\}$. (7)

Спроецированная фигура S_P задается множеством двумерных векторов спроецированных вершин $P = \{P_1,\ldots, P_N\}$, множеством индексов вершин $F = \{F_1, ..F_{3 * M}\}$, а также цветом фигуры {r, g, b, a}. Отметим, что $P_i = V_i * M_P$ (8) для всех $i \in \{1..N\}$. Три идущих подряд индекса вершин задают спроецированный полигон (треугольник).

$G_R(x, y) = \{\{r, g, b, a\}$, если существует тройка $(F_{3 * k + 1}, F3 * k + 2, F_{3 * k + 3})$, где k – целое число от 1 до M / 3, такая, что (x, y) принадлежит треугольнику, образованному вершинами P_{i1}, P_{i2}, P_{i3}, где $i1 = F_{3 * k + 1}$, $i2 = F_{3 * k + 2}$, $i3 = F_{3 * k + 3}$, {0.0, 0.0, 0.0, 0.0} в противном случае} (9).

4. МОДЕЛЬ ФИЛЬТРАЦИИ ИЗОБРАЖЕНИЙ

На основе анализа архитектуры и особенностей потоковых процессоров [4] и возможностей WebGL была разработана следующая модель фильтрации (преобразования) изображений. В основе модели лежат четыре основные понятия: Texture, Drawing Target, Filter и Filter Sequence. Texture - это изображение в форме (1), хранимое в памяти графического потокового процессора. Drawing Target - это объект, который определяет изображение-результат и цветовую маску (3).

Filter - задает преобразование изображения (2) и в общем случае представляет собой функцию с набором предопределенных и пользовательских параметров, возвращающую цвет точки для заданных координат. Функция, например, может быть задана на GLSL-подобном языке, расширенном дополнительными функциями для обработки изображений. Параметры функции строго типизированы, имеют уникальные имена и определяются описанием в форме XML. В качестве растеризуемой фигуры B используется триангулированный прямоугольник с размером равным размеру результирующего изображения R.

```
<filter>
 <function name="GetColor">
  <arguments>
   <image>buffer</image>
   <image>oldBuffer</image>
  </arguments>
  <body>
   vec4 newColor = (buffer.getOfsPixel(-1.0, -1.0) +
   buffer.getOfsPixel(-1.0,0.0) + buffer.getOfsPixel(-1.0,1.0) +
   buffer.getOfsPixel(0.0, -1.0) + buffer.getOfsPixel(0.0, 1.0) +
   buffer.getOfsPixel(1.0, -1.0) + buffer.getOfsPixel(1.0, 0.0) +
   buffer.getOfsPixel(1.0, 1.0)) / 4.0;
   vec4 oldColor = oldBuffer.getCurrentPixel();
   float clr = newColor.r - oldColor.r;
   if (clr less 0.32)
    clr = 0.32;
   if (clr more 1.0)
    clr = mod(clr, 1.0);
   return vec4(clr, clr, clr, 1.0);
  </body>
 </function>
</filter>
```

Листинг 1: Пример фильтра для расчета кадра дождевой поверхности

В листинге 1 приведен пример фильтра, заданного в предлагаемой форме. Функция GetColor должна возвратить цвет текущего обрабатываемого пикселя, аргументами функции являются два изображения buffer и oldBuffer. Функция getCurrentPixel возвращает цвет текущего пикселя в

заданном изображении, а getOfsPixel цвет пикселя со смещением относительно текущего. Типы используемых вне декларативного объявления переменных соответствуют типам данных языка GLSL.

```
<filter>
  <function name="GetColor">
  <arguments>
    <image>background</image>
    <image>rain</image>
  </arguments>
  <body>
     vec4 newColor = background.getCurrentPixel();
     vec4 oldColor = rain.getPixel(%x, %height - %y) * 0.5;
     vec4 oldColor2 = oldColor + 0.5;
          return vec4(newColor.r * oldColor2.r + oldColor.r,
newColor.g * oldColor2.g + oldColor.g, newColor.b *
oldColor2.b + oldColor.b, 1.0);
  </body>
  </function>
</filter>
```

Листинг 2: Пример фильтра для применения рассчитанной дождевой поверхности на произвольное изображение.

В листинге 2 приведен еще один пример фильтра. В этом фильтре осуществляется прямой доступ к пикселям изображения с помощью функции getPixel, а координаты рассчитываются согласно значениям предопределенных переменных %x и %y (координаты текущего пикселя), %width и %height (размеры результирующего изображения).

В основе модели фильтрации лежит условие: изображение-результат и изображения-параметры одного и того же фильтра не могут совпадать, т.е. $F(X) \neq X$. (10)

Это условие заложено в модель в целях обеспечения корректности выполнения преобразований на потоковых процессорах, т.е. для организации независимой параллельной обработки фрагментов изображения.

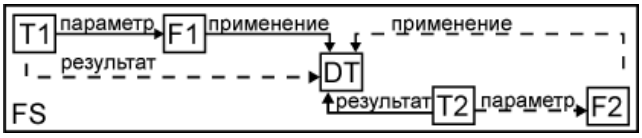

Рис.2: Взаимоотношения объектов в системе на пример filter sequence FS типа "пинг-понг".

Filter Sequence - задает последовательность нескольких фильтров с их параметрами, в общем случае позволяя организовать конвейер обработки, на основе которого можно реализовать сложное преобразование для обработки изображений. Таким образом, Filter Sequence представляет собой композицию преобразований изображений и в общем случае выражается формулой $G(X) = G_1(G_2(..G_i(X)))$, где G(X) - преобразование, задающее Filter Sequence, а G_1, G_2... G_i - составляющие его преобразования-фильтры.

На примере в рис. 2. текстура T1 является параметром фильтра F1, который применяется к drawing target DT. Текстура T2 является результатом применения фильтра F1 к DT. В тоже время текстура T2 является параметром фильтра F2, который применяется к DT. Текстура T1 является результатом применения фильтра F2 к DT. Сначала применяется F1, затем F2, исходное изображение берется из T1, результат оказывается там же. Сплошными линиями показаны связи, существующие во время применения первого фильтра, пунктирными - второго.

5. МОДЕЛЬ ПОДДЕРЖКИ ОПЕРАЦИЙ РИСОВАНИЯ

Операции рисования являются отдельным классом задач обработки изображений как при реализации на центральном процессоре, так и на потоковых. Они могут использоваться для достижения той или иной функциональности графических редакторов, например для поддержки графического комментирования материала на виртуальной доске в образовательной 3D-среде. Данные для осуществления операций могут поступать как в виде событий от клавиатуры или мыши, так и в виде данных от сервера (воспроизведение ранее записанного занятия).

Способы реализации поддержки рисования в системах обработки изображения на центральном процессоре детально проработаны. Однако, при использовании потоковых процессоров классические алгоритмы (например, алгоритм Брезентхема) оказываются неприменимы из-за архитектурных ограничений, что требует разработки методов поддержки операций рисования специально для использования с применением потоковых процессоров.

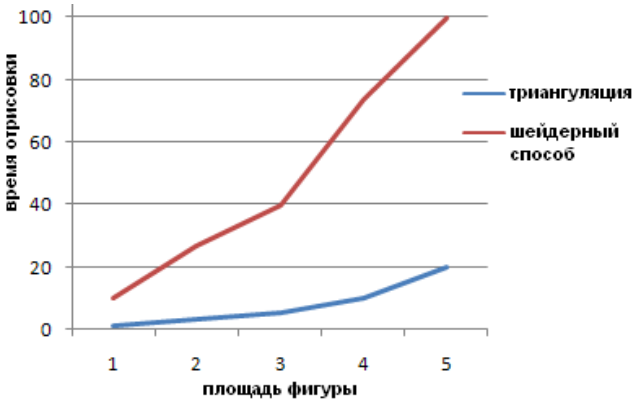

Рис. 3: Замер производительности поддержки операций рисования с помощью растеризации по аналитическому описанию и с помощью триангуляции.

Теоретически, геометрическая фигура может быть отрисована непосредственно по аналитическому описанию с применением шейдерных программ, что свело бы эту задачу к задачам фильтрации. Однако, эффективность такого подхода невелика, т.к. фактическая площадь отрисовываемой фигуры может быть на порядки меньше, чем площадь растеризовываемого триангулированного прямоугольника. Даже если в качестве данного прямоугольника брать ограничивающий объем фигуры (bounding box), неэффективность подхода остается значительной (см. рис 3).

Поэтому для более эффективной реализации необходимо триангулировать геометрические фигуры, т.е. представлять их в форме выражения (5). Триангуляцию каждой конкретной фигуры предлагается реализовать вычислением множества вершин по специфичным для выбранной фигуры формулам, т.к. триангуляция в общем виде избыточно сложна.

5.1 Программная архитектура

Программная архитектура (рис. 4) предполагает поддержку некоторой абстрактной палитры инструментов, которая может быть расширена новыми инструментами для реализации поддержки требуемых операций. Инструменты самостоятельно выполняют триангуляцию фигур и передают

на растеризацию триангулированную фигуру с установленными параметрами цвета и прозрачности.

Для каждого изображения с поддержкой операций рисования создается объект палитры инструментов, а также Drawing Target. Он содержит в себе основную текстуру, временную текстуру и буфер глубины. Размер основной и временной текстуры совпадают. Реализация текущего инструмента может запросить сохранение во временную текстуру изображения из основной текстуры перед очередным этапом работы. До окончания этапа содержимое временной текстуры может быть скопировано в основную, а поверх растеризована актуальная рисуемая фигура. Под этапом понимается полный цикл рисования одной фигуры (линии, прямоугольника и т.д).

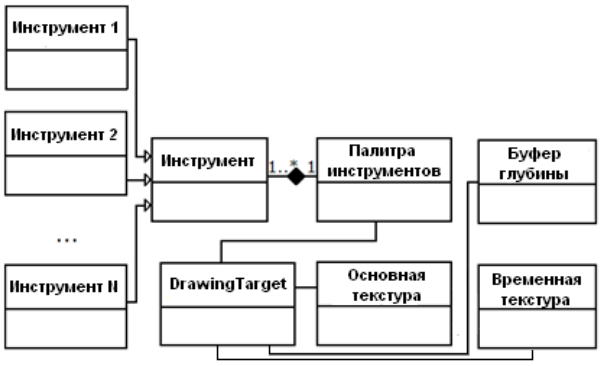

Рис. 4: Архитектура инструмента

Объект палитры инструментов содержит в себе коллекцию инструментов и хранит информацию о том, какой из инструментов активен в текущий момент. Этот объект получает уведомления о событиях, инициирующих процесс отрисовки (например, от мыши, от управляющего сервера и т.д.), и передает их текущему инструменту. Все инструменты удовлетворяют одному и тому же программному интерфейсу, так что палитра инструментов взаимодействует с любым конкретным инструментом без учета того, как этот инструмент устроен внутри.

5.2 Поддержка прозрачности

Для поддержки прозрачности предлагается использовать композицию из двух преобразований. Первый проход выражается преобразованием G(C, D, x, y) = min(C(x, y) + D(x, y), 1.0) (11) , где C и D представляют собой функции одного и того же канала изображения. Это преобразование применяется только к альфа-каналу с помощью маски (3). Оно соответствует поддерживаемому аппаратно преобразованию с параметрами source factor = GL_ONE, destination factor = GL_ONE, blend mode = GL_FUNC_ADD.

Второй проход выражается преобразованием G(A, B, x, y)q = min(A(x, y)q * B(x, y)A + (1.0 - B(x, y)A) * B(x, y)q), 1.0) (12). Преобразование (12) применяется только к RGB-каналам изображения. Оно соответствует поддерживаемому аппаратно преобразованию с параметрами source factor = GL_SRC_ALPHA, destination factor = GL_ONE_MINUS_SRC_ALPHA, blend mode = GL_FUNC_ADD. Полученное преобразование удовлетворяет предъявляемым к прозрачности требованиям:

1. Смешивание RGB-каналов должно производиться с учетом значения альфа-канала растеризуемой фигуры.
2. При растеризации фигуры на абсолютно непрозрачную точку изображения, значение альфа-канала не должно измениться.
3. При растеризации фигуры на абсолютно прозрачную точку изображения, значение альфа-канала должно увеличиться.

6. ВЫВОДЫ

Был предложен перспективный подход к обработке изображений в браузерных приложениях, который позволяет осуществлять довольно сложные преобразования в режиме реального времени. Подход успешно апробирован на ряде реальных примеров (рис. 5) и в ходе экспериментальных разработок браузерного плеера записанных виртуальных 3D-лекций (с поддержкой обработки изображений и операций рисования на виртуальной доске).

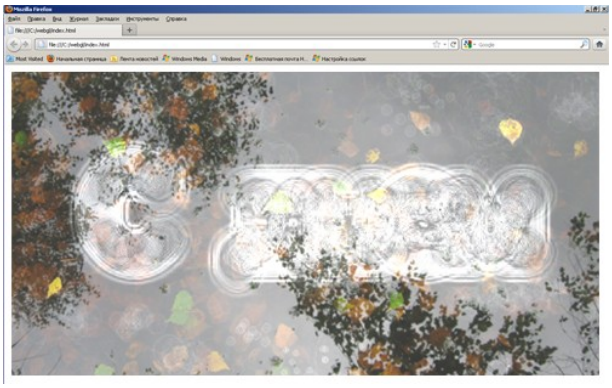

Рис. 5: Пример фильтра водной поверхности, работающего в реальном времени на потоковых процессорах в браузерном приложении в Firefox 4.

Метод показывает высокую производительность. Так box-фильтр и фрактал Мандельброта рассчитываются предлагаемым методом в ~15-110 раз быстрее, чем при использовании подхода [3], в зависимости от выбранного сочетания центрального и графического потокового процессоров.

7. ССЫЛКИ

[1] Abramoff, M.D., Magelhaes, P.J., Ram, S.J. *"Image Processing with ImageJ". Biophotonics International, volume 11, issue 7, pp. 36-42, 2004.*

[2] *Adobe Pixel Bender Guide.* http://www.adobe.com/content/dam/Adobe/en/devnet/pixelbender/pdfs/pixelbender_guide.pdf

[3] Kai Uwe Barthel, Karsten Schulz, *ImageJ in the web? - Image processing in the browser using HTML5. In proceedings of ImageJ Conference 2010.*

[4] Kayvon Fatahalian, Mike Houston. *A closer look at GPUs. Communications of the ACM, October 2008, Vol. 51, NO. 10.*

[5] The Khronos Group. *WebGL 1.0 Specification.* https://www.khronos.org/registry/webgl/specs/1.0/

[6] А.Ю. Сморкалов, М.Н. Морозов. Поддержка дискретных вейвлет-преобразований для компрессии в системе обработки изображений на потоковых графических процессорах. Материалы всероссийской научно-практической конференции "Информационные технологии в профессиональной деятельности и научной работе". - Йошкар-Ола: МарГТУ, 2010.- С. 91-96.