Совместное использование поверхностей OpenCL ™ и DirectX * 11 в графической системе процессора Intel®

Адам Лейк, Роберт Иоффе

18.02.2015

Содержание

Введение

В этом руководстве показано, как совместно использовать поверхности между OpenCl ™ и DirectX * 11 с графикой процессора Intel ® в Microsoft Windows *, используя расширение совместного использования поверхностей в OpenCL. Цель состоит в том, чтобы обеспечить доступ к выразительности, обеспечиваемой ядром OpenCL C, и возможностям рендеринга API DirectX11. Одним из примеров, где это можно было бы использовать, может быть приложение компьютерного зрения в реальном времени, которое запускает детектор признаков на изображении в OpenCL, а затем использует DirectX 11 для рендеринга окончательного вывода на экран в реальном времени с четко обозначенными функциями. Другой вариант - использовать динамически генерируемую процедурную текстуру, созданную в OpenCL, при рендеринге трехмерного объекта в сцене. Наконец, представьте себе постобработку изображения с помощью OpenCL после рендеринга сцены с помощью 3D-конвейера. Это может быть полезно для преобразования цветов, передискретизации,

Чтобы начать работу с общим доступом к поверхности, мы покажем, как обновить текстуру, созданную с помощью DirectX 11, с помощью OpenCL. Те же процессы применяются к обновлениям буфера вершин или объекта внеэкранного буфера кадра, который может использоваться в неинтерактивном автономном конвейере обработки изображений.

Расширение общего доступа к поверхности определяется в спецификации расширения OpenCL строкой
cl_khr_dx11_sharing. Мы также используем свойство контекста
CL_CONTEXT_INTEROP_USER_SYNC, который поддерживается в Intel Processor Graphics и имеет значение по умолчанию CL_FALSE.

Иногда мы используем DX11 как сокращение от DirectX 11.

Мотивация

Это руководство покажет вам, как создавать общие поверхности между OpenCL и DirectX.

Ключевые выводы

В этом руководстве показано, как разделить поверхность между OpenCL и DirectX11 с помощью
cl_khr_dx11_sharing расширение и воспользуйтесь CL_CONTEXT_INTEROP_USER_SYNC
на Intel Processor Graphics. При создании описания текстуры DX11 установите флаг MiscFlags поле
из D3D11_TEXTURE2D_DESC Возражать D3D11_RESOURCE_MISC_SHARED чтобы гарантировать отсутствие копии
происходит между OpenCL и DirectX 11. Обратите внимание, что D3D11_RESOURCE_MISC_FLAG нельзя использовать при создании ресурсов с D3D11_CPU_ACCESS флаги. Дополнительные сведения об этом последнем пункте см. В ссылках MSDN в конце статьи.

Графика процессора Intel® с общей физической памятью

Intel Processor Graphics разделяет память с CPU. На рисунке 1 показаны их отношения. Хотя это и не показано на этом рисунке, существует несколько архитектурных особенностей, расширяющих подсистему памяти. Например, иерархии кеша, сэмплеры, поддержка атомарных операций, а также очереди чтения и записи используются для получения максимальной производительности от подсистемы памяти.

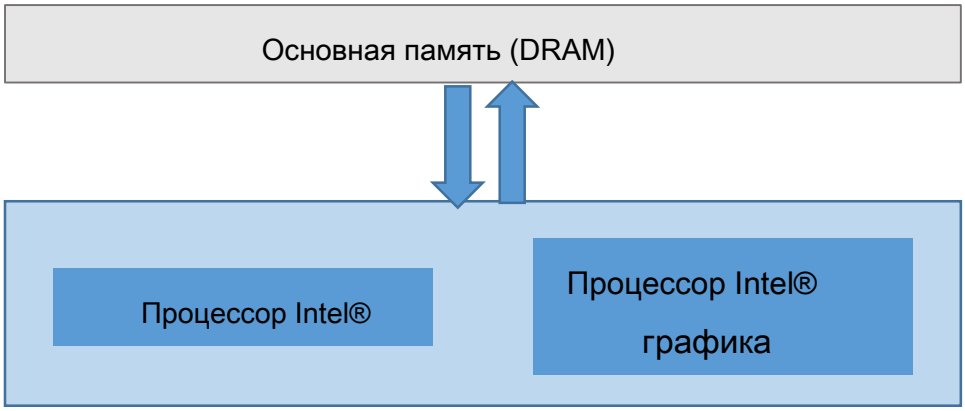


Рисунок 1. *Взаимосвязь ЦП, графики процессора Intel® и основной памяти. Обратите внимание, что один пул памяти совместно используется процессором и графическим процессором, в отличие от дискретных графических процессоров, у которых есть собственная выделенная память, которой должен управлять драйвер.*

Синхронизация между OpenCL и DirectX 11

При совместном использовании поверхностей между OpenCL и DirectX 11 важно учитывать возможность того, что программа может иметь как DirectX, так и OpenCL, пытающиеся обновить одну и ту же поверхность или расположение поверхности в одно и то же время. Есть два способа справиться с этим. Флаг CL_CONTEXT_INTEROP_USER_SYNC используется, чтобы определить, несет ли разработчик ответственность за обработку синхронизации между API. Если для этого флага установлено значение CL_FALSE, как в этом примере, драйвер будет обеспечивать выполнение операций DirectX 11 на поверхности до того, как clEnqueueAcquireD3D11ObjectsKHR отправили свои результаты в память до того, как OpenCL заработает на поверхности. Точно так же этот флаг гарантирует, что операции OpenCL завершены до того, как любые операции DX11 будут разрешены для работы на поверхности после

clEneuqueReleaseD3D11ObjectsKHR. Этот флаг и семантика упрощают совместное использование поверхностей между двумя API.

Обзор Surface Sharing между OpenCL и DirectX 11

Ниже приводится краткое изложение логики, которую пример кода выполняет для совместного использования поверхности. В следующем разделе приводится подробное объяснение каждого шага.

Инициализация

1. OpenCL:
    a. Запрос, чтобы определить, есть ли расширение cl_khr_dx11_sharing поддерживается; выйти, если неподдерживаемый.
    б. Создайте контекст, передав соответствующие параметры устройства.
    c. Создайте очередь на устройстве и контексте, который поддерживает совместное использование OpenCL и DX11.

2. DirectX: создайте текстуру DX11, которая будет использоваться OpenCL; не забудьте установить поле MiscFlags дескриптора текстуры D3D.
3. OpenCL: используя дескриптор DX11, созданный в 2, создайте общую поверхность с помощью расширения OpenCL.

Шаги 1 и 2 можно поменять местами. Шаг 3 должен следовать за шагами 1 и 2.

Запись на общую поверхность

1. Заблокируйте поверхность для монопольного доступа OpenCL.
2. Напишите на поверхность через ядро OpenCL C. В случае данных текстуры обязательно используйте функции чтения и / или записи изображения и передайте изображение соответствующим образом.
3. Разблокируйте поверхность, чтобы DirectX мог читать или записывать поверхность.

Цикл рендеринга

Цикл рендеринга использует простой проход через программируемую вершину и пиксельный шейдер для отображения текстуры двух ориентированных на экран треугольников, которые образуют четырехугольник для отображения результата. Четырехугольник использует только часть экрана, чтобы показать четкий цвет на фоне визуализации, что полезно для отладки более сложных сценариев.

Неисправность

1. Очистите объекты состояния.

Подробная информация о совместном использовании поверхностей между OpenCL и DirectX 11

Инициализация

1. OpenCL:
    a. Запрос, чтобы определить, есть ли расширение cl_khr_dx11_sharing поддерживается; выйти, если неподдерживаемый.

Не каждая реализация OpenCL поддерживает совместное использование поверхностей между OpenCL и DirectX 11, поэтому первым шагом является определение того, существует ли расширение в системе. Мы перебираем платформы, глядя на строку расширения для платформы, которая поддерживает совместное использование поверхностей. Внимательное чтение спецификации подчеркивает, что это *Платформа* расширение или *устройство* extension, поэтому мы должны проверить оба! Позже, когда мы создадим контекст, нам нужно будет запросить, какие из наших устройств в контексте могут использовать совместно с контекстом DX11.

Этот образец поддерживается только графикой процессоров Intel, но расширение его возможностей до других графических процессоров должно быть тривиальным. Строка расширения, которую мы ищем, это cl_khr_dx11_sharing. Соответствующий фрагмент кода:

```
char extension_string [1024];
memset (строка_расширения, '\ 0' , 1024);
статус = clGetPlatformInfo (платформы [i],
                                                CL_PLATFORM_EXTENSIONS ,
                                                размер (строка_расширения),
                                                extension_string,
                                                НОЛЬ );
char * extStringStart = НОЛЬ ;
extStringStart = strstr (extension_string, "cl_khr_dx11_sharing");
если (extStringStart! = 0) {
        printf («Платформа поддерживает cl_khr_dx11_sharing \ n»); …

}
```

Обратите внимание, что аналогичная последовательность кода применяется к запросу устройства и включена в пример кода на тот случай, если вы хотите изменить код для работы на других платформах.

б. Создайте контекст, передав соответствующие параметры устройства.

```
cl_context_properties cps [] =
{
        CL_CONTEXT_PLATFORM , ( cl_context_properties ) g_platformToUse,
        CL_CONTEXT_D3D11_DEVICE_KHR , ( intptr_t ) g_pd3dDevice,
        CL_CONTEXT_INTEROP_USER_SYNC , CL_FALSE ,
        0
};
```

Создайте очередь на устройстве и контексте, который поддерживает совместное использование OpenCL и DX11. Обратите внимание, что в контексте мы указали флаг CL_CONTEXT_INTEROP_USER_SYNC и установили его в CL_FALSE, что является значением по умолчанию. Используйте этот флаг, чтобы указать, собираетесь ли вы управлять синхронизацией между OpenCL и DX11 или позволить среде выполнения обрабатывать это.

Сначала мы запрашиваем количество устройств на нашей платформе, которые соответствуют нашим критериям. Обратите внимание, поскольку это расширение, и мы уже проверили, поддерживает ли платформа это расширение, мы должны создать указатель на функцию расширения, которую мы используем на этом следующем шаге.

```
clGetDeviceIDsFromD3D11KHR_fn ptrToFunction_clGetDeviceIDsFromD3D11KHR = НОЛЬ ;
ptrToFunction_clGetDeviceIDsFromD3D11KHR = ( clGetDeviceIDsFromD3D11KHR_fn )
clGetExtensionFunctionAddressForPlatform (g_platformToUse, "clGetDeviceIDsFromD3D11KHR" );

cl_uint numDevs = 0;
// осторожно с g_pd3DDevice
status = ptrToFunction_clGetDeviceIDsFromD3D11KHR (g_platformToUse,
CL_D3D11_DEVICE_KHR , ( пустота *) g_pd3dDevice, CL_PREFERRED_DEVICES_FOR_D3D11_KHR , 0,
НОЛЬ , & numDevs);
testStatus (статус, "Ошибка clGetDeviceIDsFromD3D11KHR" );
```

Используя эту информацию, мы создаем устройство только с контекстом DX11, который мы просили предоставить.

```
cl_device_id * devID = NULL;
g_clDevices = (cl_device_id *) malloc ( размер (cl_device_id) * numDevs); ptrToFunction_clGetDeviceIDsFromD3D11KHR
(g_platformToUse, CL_D3D11_DEVICE_KHR ,
( пустота *) g_pd3dDevice, CL_PREFERRED_DEVICES_FOR_D3D11_KHR , numDevs, g_clDevices, NULL); testStatus (статус, "Ошибка
clGetDeviceIDsFromD3D11KHR" );

// создаем контекст OCL из устройства, которое мы используем в качестве устройства рендеринга DX11
g_clContext = clCreateContext (cps, 1, g_clDevices, NULL, NULL, & статус); testStatus (статус, «Ошибка
clCreateContext» );
```

Наконец, создайте нашу очередь команд для приложения на этом устройстве. Обратите внимание, что это нацелено на то, чтобы совместное использование поверхностей хорошо работало на графике процессора Intel, поэтому мы собираемся избавить от некоторых сложностей, связанных с созданием переносимого кода платформы для выбора устройства.

```
// создаем очередь команд openCL
g_clCommandQueue = clCreateCommandQueue (g_clContext, devID, 0 и статус); testStatus (статус, «Ошибка
clCreateCommandQueue» );
```

2. DirectX: создайте текстуру DX11, которой мы поделимся с OpenCL. Здесь мы используем флаг D3D11_RESOURCE_MISC_SHARED, чтобы гарантировать создание оптимального сценария совместного использования с Intel Processor Graphics.

```
пустота CreateTextureDX11 ()
{
	беззнаковый символ * текстура = НОЛЬ ; текстура = ( беззнаковый символ *) malloc ( размер ( беззнаковый символ ) * NUM_IMAGE_CHANNELS
	*
SHARED_IMAGE_HEIGHT * SHARED_IMAGE_WIDTH );
	если (текстура == nullptr )
	{
		printf ( "ошибка создания текстуры \ n" );
	}

	за ( беззнаковое целое я = 0; я < NUM_IMAGE_CHANNELS * SHARED_IMAGE_HEIGHT *
SHARED_IMAGE_WIDTH ;)
	{
		текстура [i ++] = 255;
		текстура [i ++] = 0;
		текстура [i ++] = 0;
		текстура [i ++] = 255;
	}

	D3D11_TEXTURE2D_DESC desc;
	ZeroMemory (& desc, размер ( D3D11_TEXTURE2D_DESC ));
	desc.Width = SHARED_IMAGE_WIDTH ;
	desc.Height = SHARED_IMAGE_HEIGHT ;
	desc.MipLevels = 1;
	desc.ArraySize = 1;
	desc.Format = DXGI_FORMAT_R8G8B8A8_UNORM ; desc.SampleDesc.Count = 1; desc.SampleDesc.Quality
	= 0;
	desc.Usage = D3D11_USAGE_DEFAULT ;
	desc.BindFlags = D3D11_BIND_SHADER_RESOURCE ;
	desc.CPUAccessFlags = 0;
	если (g_UseD3D11_RESOURCE_MISC_SHAREDflag == true)
	{
		printf ("Использование флага D3D11_RESOURCE_MISC_SHARED \ n"); desc.MiscFlags
		= D3D11_RESOURCE_MISC_SHARED;
	}
	еще
	{
		desc.MiscFlags = 0;

	}
	D3D11_SUBRESOURCE_DATA столовые ложки;
	ZeroMemory (& tbsd, размер ( D3D11_SUBRESOURCE_DATA ));
	tbsd.pSysMem = ( пустота *) текстура; tbsd.SysMemPitch = SHARED_IMAGE_WIDTH * NUM_IMAGE_CHANNELS
	; tbsd.SysMemSlicePitch = SHARED_IMAGE_WIDTH * SHARED_IMAGE_HEIGHT *

NUM_IMAGE_CHANNELS ;
	g_pd3dDevice-> CreateTexture2D (& desc, & tbsd, & g_pSharedDX11Texture2D);
	// еще нужно привязать

	бесплатно (текстура);
}
```

3. OpenCL: используя дескриптор DX11, созданный в 2, создайте общую поверхность с помощью расширения OpenCL.

Это сердце нашего API общего доступа к поверхностям. Совместное использование будет работать, только если следующий вызов завершится успешно:

```
int ПоделитьсяDX11BufferWithCL ()
```

```
{
        int статус = 0;

        g_SharedRGBAimageCLMemObject =
ptrToFunction_clCreateFromD3D11Texture2DKHR (g_clContext, CL_MEM_WRITE_ONLY , g_pSharedDX11Texture2D, 0,
& статус);
        если (статус == 0)
        {
                printf ( "Успешно предоставлен доступ! \ N" );
                статус = УСПЕХ ;
        }
        еще
        {
                printf ( "Не удалось отправить \ n" );
                статус = ПОТЕРПЕТЬ ПОРАЖЕНИЕ ;
        }
        возвращение положение дел;
}
```

Шаги 1 и 2 можно поменять местами. Шаг 3 должен следовать за шагами 1 и 2.

Запись на общую поверхность

Нам необходимо убедиться, что когда OpenCL читает или записывает поверхность, DX11 не использует поверхность, и наоборот. Для этого расширение поддерживает функции получения и освобождения монопольного доступа к поверхности. Здесь мы описываем шаги по использованию этих API. Есть два способа сделать это. Во-первых, мы могли использовать в нашем коде объекты синхронизации. Во-вторых, и то, как мы делаем это в этом примере, заключается в том, чтобы воспользоваться преимуществом поведения, когда мы используем флаг CL_CONTEXT_INTEROP_USER_SYNC установлен в CL_FALSE. Когда этот флаг установлен, синхронизация неявна в clEnqueueAcquireD3D11ObjectsKHR () и

clENqueueReleaseD3D11ObjectsKHR () Стороны вызова API.

4. Заблокируйте поверхность для монопольного доступа OpenCL.

```
status = ptrToFunction_clEnqueueAcquireD3D11ObjectsKHR (g_clCommandQueue, 1, &
g_SharedRGBAimageCLMemObject, 0, 0, 0);
```

5. Пишите на поверхность через ядро OpenCL C. В случае данных текстуры обязательно используйте image читают и / или записывают функции и передают изображение соответствующим образом. Ядро просто записывает в подмножество текселей текстуры.

```
ядро недействительно drawBox ( __write_only image2d_t вывод, плавать fDimmerSwitch) {

        int x = get_global_id (0);
        int y = get_global_id (1);

        int xMin = 0, xMax = 1, yMin = 0, yMax = 1;

        если ((x> = xMin) && (x <= xMax) && (y> = yMin) && (y <= yMax)) {

                write_imagef (вывод, ( int2 ) (x, y), ( float4 ) (0.f, 0.f, fDimmerSwitch, 1.f));
        }
}
```

6. Разблокируйте поверхность, чтобы OpenCL теперь мог читать или записывать поверхность.

status = ptrToFunction_clEnqueueReleaseD3D11ObjectsKHR (g_clCommandQueue, 1, &
g_SharedRGBAimageCLMemObject, 0, НОЛЬ , НОЛЬ);

Цикл рендеринга

Цикл рендеринга использует простой проход через программируемую вершину и пиксельный шейдер для отображения текстуры двух ориентированных на экран треугольников, образующих четырехугольник для отображения результата.

Неисправность

1. Очистите государственные объекты.

Этот образец не требует много работы для очистки. Единственный объект - это cl_mem объект для общей поверхности. На стороне DX11 мы должны освободить текстуру, которую мы создали для публикации, сэмплер для текстуры и вид текстуры.

Будущая работа

В этом руководстве рассматриваются основы совместного использования поверхностей. Имея больше времени, мы хотели бы расширить рамки учебника, чтобы охватить дополнительные варианты использования, затронутые здесь.

Совместное использование явных событий синхронизации между OpenCL и DirectX 11

В дополнение к неявной синхронизации, используемой в этой статье, OpenCL и DirectX 11 также могут использовать
Флаг CL_CONTEXT_INTEROP_USER_SYNC установлен в CL_TRUE и обрабатывать синхронизацию
явно с объектами синхронизации Windows. Было бы интересно сравнить производительность двух. Архитекторы драйверов, с которыми мы говорили, обнаружили, что разница минимальна или отсутствует вообще, поэтому это текущая рекомендация, но было бы поучительно проверить.

Совместное использование буферов кадра, глубины, трафарета и поверхностей MSAA

В OpenCL и DirectX 11 отсутствуют возможности совместного использования поверхностей для многих полезных поверхностей. Было бы полезно, чтобы они поддерживались аналогично случаю с совместным использованием поверхностей между OpenGL и DirectX 11. Кроме того, хотя расширение и документация не содержат явных указаний об этом, с этим расширением требуется поддерживать только совместное использование поверхностей без мипмаппинга.

Двойная буферизация

Мы могли бы изучить компромиссы в сложности и производительности, используя схему двойной буферизации. В этом руководстве мы сосредоточились на функциональности и основах совместного использования поверхностей.

Что делать, если совместное использование поверхности не поддерживается?

У Максима Шевцова есть статья, цитируемая в справочных материалах, в которой описывается случай, когда необходимо копирование между OpenCL и OpenGL. Хотя эти рекомендации ориентированы на OpenGL, многие из рекомендаций относятся к совместному использованию с DirectX 11.

Пример совместного использования поверхности

Зависимости

Образец совместного использования поверхности имеет следующие зависимости:

- SDK для Windows 8 или выше; обратите внимание, что DirectX SDK теперь устарел, а Windows SDK используется для разработки Direct3D *.

- Microsoft Visual Studio * 2012 или 2013
    - Используйте части SDK, относящиеся к DirectX, включая d3dcompiler_46.dll. D3dcompiler_46.dll копируется в
    - соответствующий каталог Debug или Release, в котором исполняемые файлы находятся на уровне РЕШЕНИЯ (а не ПРОЕКТА). Есть и другие способы справиться с этой зависимостью.

    - Этот образец должен работать с другими версиями Visual Studio, но не тестировался, Intel® INDE (Intel® OpenCL ™
- Code Builder сейчас является его частью):
    - Использовать cl.h, cl_d3d11.h, и opencl.lib файлы

В образце использовались следующие настройки, но ваши могут немного отличаться:

- Копировать d3dcompiler_46.dll в каталог выпуска отладчика, в котором будут построены исполняемые файлы на уровне РЕШЕНИЯ (не ПРОЕКТА).
- Добавьте местоположение cl.h и cl_d3d11.h на ваш путь включения. Например,
    - C: \ Program Files (x86) \ Intel \ OpenCL SDK \ 3.0 \ include \ CL
- Добавьте расположение библиотеки OpenCL к пути к вашей библиотеке: например,
    - C: \ Program Files (x86) \ Intel \ OpenCL SDK \ 3.0 \ lib \ x86
- Добавить OpenCL.lib в ваш набор статически связанных библиотек.

Пример структуры файлов и каталогов

Решение находится в каталоге CL_20_DX11_surface_sharing. В каталоге с тем же именем находятся файлы проекта и исходные файлы. Этот пример кода ориентирован на демонстрацию открытого обмена между OpenCL и DirectX 11 и не предназначен для качественной реализации продукта.

Файлы организованы следующим образом:

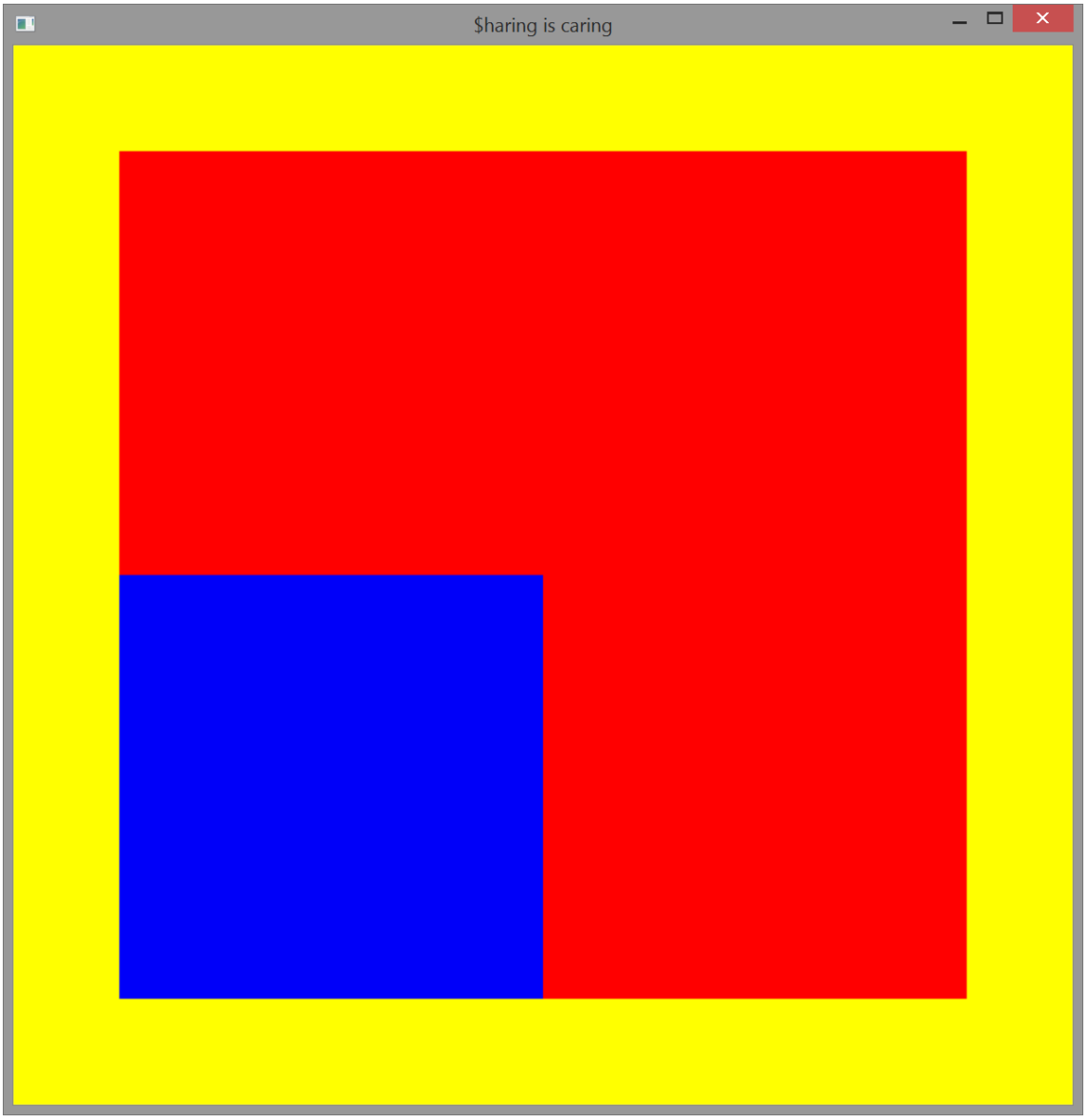
- Main.cpp - В основном оконный код, цикл перекачки сообщений Windows, отправляет события в соответствующие функции OpenCL и DX11 и обработчик клавиатуры
- Common_CL_20_DX11.h - Флаги, которые используются как для OpenCL, так и для DX11
- DX11.h, DX11.cpp - Функции, характерные для DX11, включая инициализацию, выключение и рендеринг

- OCL.h, OCL.cpp - Функции, специфичные для OpenCL
- OpenCLRGBAFile.cl - Исходный код ядра OpenCL
- PixelShader.hlsl, VertexShader.hlsl - Простые вершинные и пиксельные шейдеры для DX11

Сборка и запуск примера

Создайте этот образец кода, выбрав *Сборка-> Сборка решения* из главного меню. Все исполняемые файлы должны быть сгенерированы. Вы можете запустить их напрямую в Visual Studio или перейти в каталоги Debug и / или Release, которые находятся в том же месте, что и каталог. CL_20_DX11_surface_sharing файл решения.

Чтобы запустить образец, нажмите F5 в интегрированной среде разработки Visual Studio. Три соответствующих файла ядра и шейдера: OpenCLRGBAFile.cl, PixelShader.hlsl, и VertexShader.hlsl.



Фигура 2. *Ожидаемый результат выборки. Желтый - это чистый цвет фона. Изображение представляет собой четырехугольник с ориентацией на экран, состоящий из двух треугольников с наложенной текстурой. Текстура представляет собой небольшую красную текстурную карту 4x4 с нижней левой частью текселей, записываемых OpenCL ™ после первоначального заполнения OpenGL *. OpenCL записывает значение в канал синего цвета, меняя цвет от черного до синего (от 0 до 255 в синем канале).*

Узнать больше

Существуют дополнительные типы буферов, которые вы можете использовать совместно с OpenCL и DirectX 11. Кроме того, вы можете использовать дополнительные механизмы синхронизации. Они подробно описаны в спецификации расширения OpenCL, и вы можете найти там более подробную информацию.

Благодарности

Спасибо Мурали Сундарасену, Аарону Кунце, Аллену Хаксу, Павану Ланке, Максиму Шевцову, Михалу Мрозеку, Петру Умински, Стивену Джанкинсу, Дэну Петре и Бену Эшбо. Все они были доступны для технических обсуждений, уточнений или обзоров в процессе.

Рекомендации

1. Спецификация OpenCL 1.2: https://www.khronos.org/registry/cl/
2. Спецификация OpenCL 2.0, состоящая из трех томов: спецификация языка OpenCL C, API среды выполнения OpenCL и расширения OpenCL: https://www.khronos.org/registry/cl/
3. Технический документ Стивена Джанкинса: Вычислительная архитектура графической системы для процессоров Intel® Gen 7.5: https://software.intel.com/sites/default/files/managed/f3/13/Compute_Architecture_of_Intel_Proc essor_Graphics_Gen7dot5_Aug2014.pdf . Это обязательное чтение для всех, кто использует OpenCL на графических платформах процессоров Intel®.
4. Учебник Адама Лейка по минимизации буферных копий на Intel Processor Graphics: https://software.intel.com/en-us/articles/getting-the-most-from-opencl-12-how-to-increaseperformance-by-minimizing-buffer-copies-on-intel-pr
5. Объекты синхронизации в Windows: http://msdn.microsoft.com/enus/library/windows/desktop/ms686364(v=vs.85).aspx
6. Ограничения использования флага D3D11_RESOURCE_MISC_FLAG с D3D11_CPU_ACCESS: http://msdn.microsoft.com/en-us/library/windows/desktop/ff476203(v=vs.85).aspx
7. Ограничения на типы поверхностей, которые могут поддерживать совместное использование поверхностей: http://msdn.microsoft.com/en-
us / library / windows / desktop / hh404562 (v = vs.85) .aspx # direct3d_device_sharing

Определения

Ниже приведены определения некоторых терминов, используемых в этом руководстве. Для получения более подробной информации обратитесь к ссылкам.

- **Буферы:** OpenCL различает буферы и изображения. Буферы OpenCL выложены *линейно* в памяти - думайте о буфере как о массиве.
- **Текстуры:** Обратитесь к буферам данных, размещенным в *выложенный плиткой* форматировать и считывать через встроенные семплеры в OpenGL или DX11. Такая структура памяти позволяет повысить производительность за счет сэмплеров текстур, которые фильтруют входные пиксели, считываемые из памяти, через предварительно заданные ядра фильтров.
- **Поверхность:** Относится к буферам, текстурам или изображениям. Это общий термин для данных в памяти, макет которых может быть мозаичным или линейным. В некоторых случаях поверхность имеет дополнительные данные, такие как размер, высота, ширина и атрибуты компоновки данных. Эти атрибуты управляются через API (OpenCL, OpenGL, DirectX и т. Д.).
- **Сэмплеры:** Используются для чтения изображений в OpenCL и текстур в OpenGL или DX11. Сэмплер использует внутренние кеши и мозаичный макет изображения или текстуры в памяти для повышения производительности при фильтрации. Сэмплеры включают в себя кеши и логику для выполнения выборки из нескольких текселей и (возможно) уровней MIP-карты одновременно и вывода одного значения текселя для одного запроса.

- **Картинки:** Относится к буферам данных, размещенным в мозаичном формате и считываемым через встроенные семплеры в OpenCL. Они эквивалентны текстурам OpenGL или DX11. Какие форматы изображений и форматы текстур могут использоваться совместно или поддерживаться, зависит от конкретной реализации.
- **Совместное использование поверхности:** Сокращение для *совместное использование поверхностей между API* и используется для обозначения создания поверхности в одном API и использования данных в другом. Мотивация состоит в том, чтобы свести к минимуму создание нескольких копий одной и той же поверхности, но это не совсем так, если мы не соблюдаем набор ограничений, зависящих от устройства. В этом руководстве описаны эти ограничения для Intel Processor Graphics.
- **Отображение текстуры:** Связь пикселей в памяти с многоугольником в графическом конвейере. В этом примере мы отображаем текстуру OpenGL или DX11 на два ориентированных на экран многоугольника для отображения.

- **Общая физическая память:** Хост и устройство используют одну и ту же физическую память DRAM. Это отличается от общего *виртуальный* память, когда хост и устройство используют одни и те же виртуальные адреса, и это не является предметом данной статьи. Ключевой аппаратной функцией, обеспечивающей совместное использование поверхности, является то, что ЦП и ГП совместно используют *физический* объем памяти. Общая физическая и общая виртуальная память не исключают друг друга. Устройства могут не видеть всю физическую память для поддержки общей физической памяти.

- **Графика процессора Intel®:** Термин, используемый в отношении текущих графических решений Intel. Названия продуктов для графических процессоров Intel, интегрированных в SoC, включают графику Intel® Iris ™, графику Intel® Iris ™ Pro или графику Intel® HD Graphics в зависимости от конкретной SoC. Дополнительные сведения об аппаратной архитектуре см. В документе «Вычислительная архитектура графического процессора Intel® Gen 7.5», указанном в разделе «Ссылки» этого документа, или http://ark.intel.com/ .

Легальная информация

ИНФОРМАЦИЯ В ДАННОМ ДОКУМЕНТЕ ПРЕДОСТАВЛЯЕТСЯ В СВЯЗИ С ПРОДУКТАМИ INTEL. ДАННЫЙ ДОКУМЕНТ НЕ ПРЕДОСТАВЛЯЕТ НИКАКИХ ПРАВ НА ИНТЕЛЛЕКТУАЛЬНУЮ СОБСТВЕННОСТЬ, ЯВНЫХ ИЛИ ПОДРАЗУМЕВАЕМЫХ, ОТ ESTOPPEL ИЛИ Иным образом. ЗА ИСКЛЮЧЕНИЕМ УСЛОВИЙ ПРОДАЖИ ТАКИХ ПРОДУКТОВ INTEL, INTEL НЕ НЕСЕТ НИКАКОЙ ОТВЕТСТВЕННОСТИ И ОТКАЗЫВАЕТСЯ ОТ ЛЮБЫХ ЯВНЫХ ИЛИ ПОДРАЗУМЕВАЕМЫХ ГАРАНТИЙ В ОТНОШЕНИИ ПРОДАЖИ И / ИЛИ ИСПОЛЬЗОВАНИЯ ГАРАНТИЙНЫХ ПРОДУКТОВ, ВКЛЮЧАЯ ОТВЕТСТВЕННОСТЬ ПО ОТВЕТСТВЕННОСТИ ПО ОТВЕТСТВЕННОСТИ КОММЕРЧЕСКАЯ ЦЕННОСТЬ ИЛИ НАРУШЕНИЕ ЛЮБОГО ПАТЕНТА, АВТОРСКОГО ПРАВА ИЛИ ДРУГОГО ПРАВА НА ИНТЕЛЛЕКТУАЛЬНУЮ СОБСТВЕННОСТЬ.

«Критически важное приложение» - это любое приложение, в котором отказ продукта Intel может прямо или косвенно привести к травмам или смерти. ПРИ ПОКУПКЕ ИЛИ ИСПОЛЬЗОВАНИИ ПРОДУКЦИИ INTEL ДЛЯ ЛЮБОГО ТАКОГО КРИТИЧЕСКОГО ПРИМЕНЕНИЯ ВЫ ОБЕСПЕЧИВАЕТЕ КОМПЕНСАЦИЮ И ДЕРЖАТЬ INTEL И ЕЕ ДОЧЕРНИЕ ПРЕДПРИЯТИЯ, СУБПОДРЯДЧИКОВ И АФФИЛИРОВАННЫХ ЛИЦ, ДИРЕКТОРОВ, ДОЛЖНОСТНЫХ ЛИЦ И СОТРУДНИКОВ, СОТРУДНИКОВ И СОТРУДНИКОВ ЕГО РАЗУМНЫЕ ВОЗВРАЩЕНИЯ АДВОКАТОВ, ВОЗНИКАЮЩИЕ НАПРЯМУЮ ИЛИ КОСВЕННО ЛЮБОЙ ПРЕТЕНЗИИ ОБ ОТВЕТСТВЕННОСТИ ЗА ПРОДУКЦИЮ, ЛИЧНЫХ ТРАВМАХ ИЛИ СМЕРТИ, ВОЗНИКАЮЩИХ ЛЮБОЙ СПОСОБОМ ИЗ ТАКОЙ ЗАДАЧИ КРИТИЧЕСКОЕ ПРИЛОЖЕНИЕ, БЫЛА ЛИБО ИЛИ НЕ УСТРАИВАЛА ИЛИ НЕ УСТРАИВАЛА СВОЕ ПОДРАЗДЕЛЕНИЕ WASCON ПРЕДУПРЕЖДЕНИЕ ПРОДУКТА INTEL ИЛИ ЛЮБОЙ ЕГО ЧАСТИ.

Intel может вносить изменения в спецификации и описания продуктов в любое время без предварительного уведомления. Разработчики не должны полагаться на отсутствие или характеристики каких-либо функций или инструкций, помеченных как «зарезервировано» или «не определено». Intel резервирует их для будущего определения и не несет никакой ответственности за конфликты или несовместимость, возникающие в результате будущих изменений в них. Информация здесь может быть изменена без предварительного уведомления. Не завершайте дизайн этой информацией.

Продукты, описанные в этом документе, могут содержать конструктивные дефекты или ошибки, известные как дефекты, которые могут привести к отклонению продукта от опубликованных спецификаций. Текущие исправления доступны по запросу.

Обратитесь в местное торговое представительство Intel или к своему дистрибьютору, чтобы получить последние спецификации и перед размещением заказа на продукцию.

Копии документов с порядковыми номерами, на которые есть ссылки в этом документе или в другой литературе Intel, можно получить, позвонив по телефону 1-800-548-4725, или перейдите по адресу: http://www.intel.com/design/literature .htm

Программное обеспечение и рабочие нагрузки, использованные в тестах производительности, могли быть оптимизированы для работы только на микропроцессорах Intel. Тесты производительности, такие как SYSmark * и MobileMark *, проводятся с использованием определенных компьютерных систем, компонентов, программного обеспечения, операций и функций. Любое изменение любого из этих факторов может привести к изменению результатов. Вам следует ознакомиться с другой информацией и тестами производительности, чтобы помочь вам в полной мере оценить ваши предполагаемые покупки, включая характеристики этого продукта в сочетании с другими продуктами.

Intel, логотип Intel, Iris и Iris Pro являются товарными знаками Intel Corporation в США и / или других странах.

* Другие названия и бренды могут быть заявлены как собственность других лиц.

OpenCL и логотип OpenCL являются товарными знаками Apple Inc., используемыми с разрешения Khronos.