

GPU Computing

GeForce and Radeon OpenCL Test

Publicado em 15.jan.2010 em www.geeks3d.com - Tradução por Luiz Gustavo TURATTI

<http://www.geeks3d.com/20100115/gpu-computing-geforce-and-radeon-opencl-test-part-1/>

<http://www.geeks3d.com/20100115/test-gpu-computing-geforce-and-radeon-opencl-test-part-2/>

<http://www.geeks3d.com/20100116/test-gpu-computing-geforce-and-radeon-opencl-test-part-3/>

<http://www.geeks3d.com/20100119/test-gpu-computing-geforce-and-radeon-opencl-test-part-4-and-conclusion/>

Aqui estão os resultados dos testes de desempenho do OpenCL com as placas da NVidia e AMD/ATI. Foram utilizadas as demonstrações (demos) OpenCL do programa **GPU Caps Viewer 1.8.2** [1].

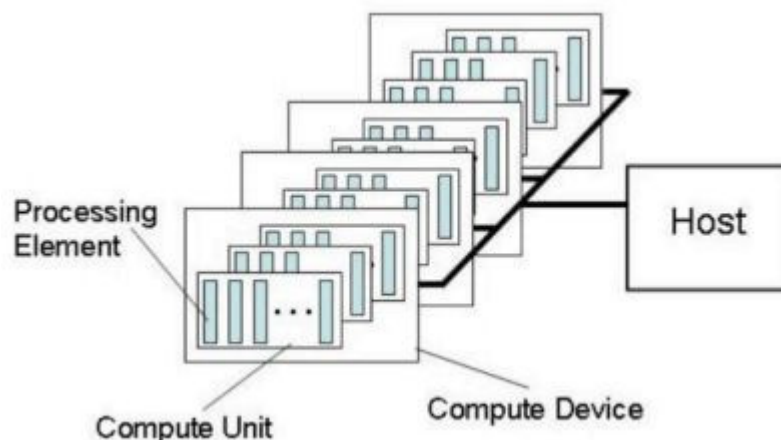
Cada “demo” está disponível para: CPU e GPU. Atualmente, nesta versão do programa, os exemplos para CPU funciona apenas na plataforma AMD, uma vez que AMD é o único fabricante a prover a implementação para CPU no OpenCL. Em contrapartida, a implementação para GPU é suportada pela NVIDIA e AMD. Este artigo é focado no **OpenCL GPU code path**.

OpenCL

OpenCL (Open Computing Language) [2] é um padrão aberto para programação paralela de propósito geral entre diversos processadores: CPUs, GPUs e NPU's (Network Processing Unit). É uma API e uma linguagem. A API (Application Programming Interface) é usada para gerenciar as entidades (dispositivos, contexto, kernels, entre outras) enquanto a linguagem é a linguagem de programação (baseada em C) usada para escrever kernels.

O OpenCL oferece as mesmas funcionalidades do NVidia CUDA ou Microsoft DirectCompute. Todas essas tecnologias permitem utilizar o poder computacional das GPUs modernas que são altamente paralelizadas para resolver problemas de propósito geral. Uma característica do OpenCL é que esta linguagem não depende de outras APIs ou tecnologias. É possível utilizar apenas OpenCL numa simples linha de comando (veja a explicação de Neil Trevett [3]) para realizar alguns cálculos pesados.

Uma plataforma OpenCL é composta por um ou vários dispositivos computacionais. Um dispositivo computacional é uma GPU, por exemplo. Atualmente todos os exemplos do programa GPU Caps usam apenas um dispositivo computacional. Caso possua uma placa GeForce GTX295, apenas uma GPU será usada. Cada dispositivo computacional possui diversas unidades computacionais, que possuem diversos elementos de processamento (ALU, cache, memória compartilhada). Já um kernel é um programa escrito na linguagem OpenCL que é executado por todas as unidades computacionais ao mesmo tempo, que trata o fluxo de dados multiprocessado (SIMD).



Modelo da plataforma OpenCL

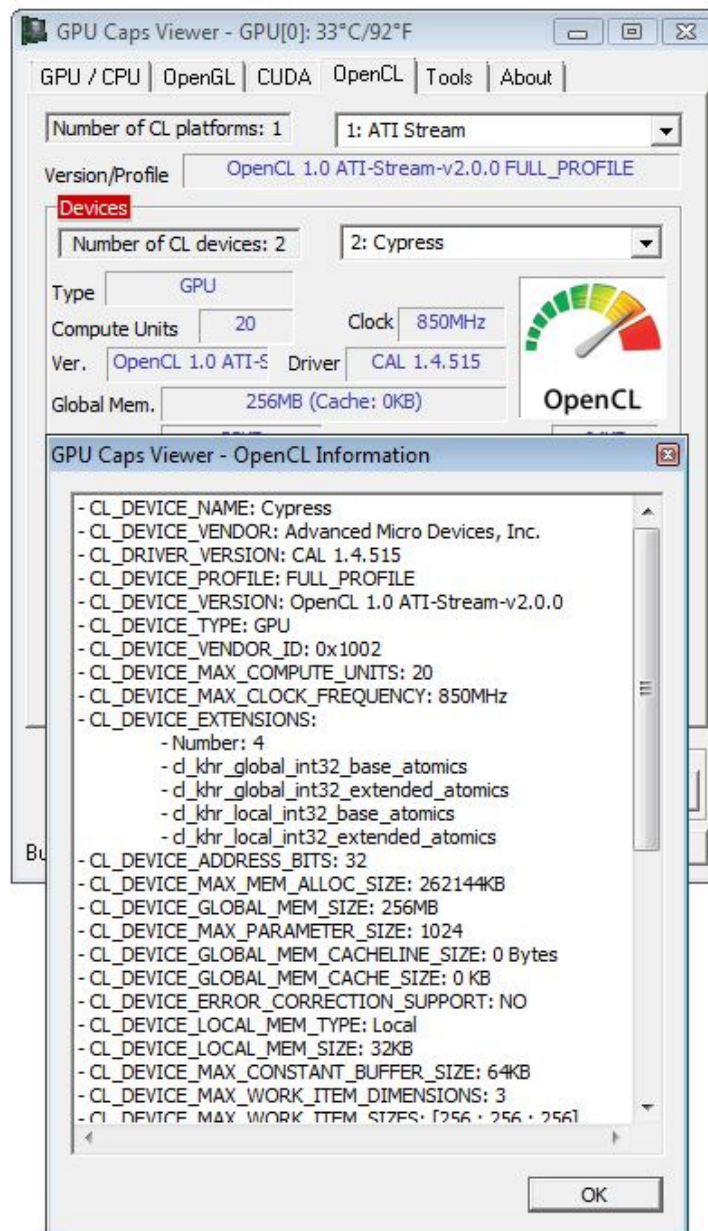
Vejamos alguns dados numéricos sobre a plataforma, baseados nas definições apresentadas:

Placa	Modelo	Compute Units	Processors	OpenCL Processing Elements
NVIDIA GeForce	GTS 250	16	128	8 (128/16)
	GTX 280	30	240	8 (240/30)
ATI Radeon	HD 5670 [4]	5 (SIMD)	400	80 (400/5)
	HD 5770	10	800 (160 vec5)	80 (800/10)
	HD 5870 [5]	20	1600 (320 vec5)	80 (1600/20)

Uma Radeon da série HD5000 possui 80 elementos de processamento (16 núcleos de processamento com 5 ALUs por núcleo) enquanto uma GeForce possui apenas 8 elementos de processamento.

A programação de um aplicativo em OpenCL é similar a programação em OpenGL, onde são necessárias bibliotecas (headers, libs e DLLs) ofertadas pela NVIDIA [6] e AMD juntamente com os drivers dessas placas. Onde encontrar exemplos de OpenCL? A NVIDIA, AMD e Apple possuem seus próprios kits de desenvolvimento com diversos exemplos neles.

Para verificar a capacidade de sua placa, veja as informações que o programa GPU Caps Viewer apresenta, como no exemplo a seguir:

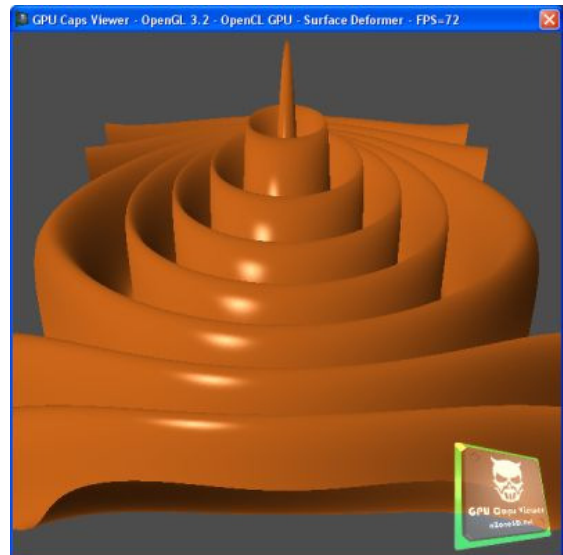


Primeiro Teste: Deformação de Superfície

A deformação de superfície foi o primeiro demo codificado em OpenCL, com base no exemplo simpleGL da NVIDIA OpenCL SDK, melhorado para renderizar uma malha real com luminosidade ao invés de uma grade plana de vértices coloridos.

Inicialmente a demonstração foi criada numa **GeForce GTS 250** e na primeira versão foi usada uma malha plana de 800x800 vértices (cerca de 1200 milhões de triângulos). A deformação funcionava a cerca de **6 FPS**. Já numa GTX 295 foi obtido 14 FPS. Quando testado numa **Radeon HD 5770** obtive uma grande diferença: **51 FPS!!!**

Existem duas possíveis explicações: as placas Radeon HD5000 são muito, muito poderosas ou existem problemas no driver OpenCL da NVIDIA. Na verdade, ambas explicações fazem sentido.



Demonstração de malha com 512x512 vértices

O problema existente no driver da NVIDIA estava na versão ForceWare 190.89. Uma atualização simples para a versão 195.39 [7] foi suficiente para melhorar o desempenho em 12 FPS na GTS 250 e para 21 FPS na GTX 295. E com a última versão do NVIDIA WHQL R195.62 [8] a performance da NVIDIA finalmente alcançou os resultados da AMD. Então, todos os testes aqui realizados foram feitos com o R195.62.

Mas é interessante observar as razões da baixa performance da NVIDIA com os primeiros drivers OpenCL. Após alguns testes, leituras e discussões, foram encontradas duas coisas para otimizar: **funções transcendentais** (seno, cosseno, ...) que são funções processadas pela SFU (Special Functions Unit) e **tamanho do grupo de trabalho** em OpenCL.

Em OpenCL são definidos três tipos de funções matemáticas para `seno` ou `sqrt`:

- Um exemplo de função regular: $y = \sin(x)$;
- Um exemplo de função nativa: $y = \text{native_sin}(x)$;
- Um exemplo de função mediana: $y = \text{half_sin}(x)$;

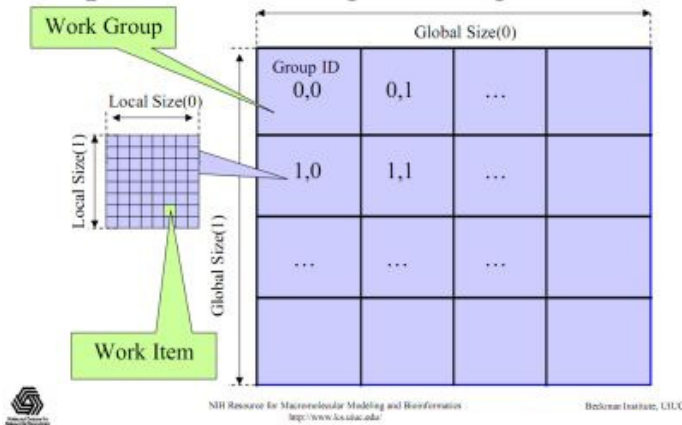
Observando as especificações do OpenCL: “A subset of functions from table 6.7 that are defined with the `native_` prefix. These functions may map to one or more native device instructions and will typically have better performance compared to the corresponding functions (without the `native_` prefix) described in table 6.7. The accuracy (and in some cases the input range(s)) of these functions is implementation-defined.”

Assim, notamos que na GPU GeForce as funções transcendentais são processadas por duas SFUs ou unidades computacionais em OpenCL enquanto na GPU Radeon as funções transcendentais são processadas por uma unidade especial em cada processador `vec5`. Na GTS 250, o uso de funções nativas melhorou o desempenho de 12 para 19 FPS. Isto foi bom, mas pode ser melhorado com **explicit work group size**.

Existem duas maneiras de executar um código OpenCL: **explicitamente**, especificando o tamanho do grupo (global e local) ou deixando o driver OpenCL fazer seu trabalho (**implicitamente**, apenas global). O **work group size** no OpenCL equivale ao **block thread size** do CUDA. Esta especificação é chamada de **NDRange configuration** [9].

Para a placa GeForce GTS 250 com driver R195.39, quando utilizadas especificações explícitas para o tamanho do grupo de trabalho, tivemos um impacto muito importante em FPS. Na ATI Radeon série HD5000 não houve impacto perceptível com uso de explicit work group size ou funções `native_`. Quando o driver da NVIDIA foi atualizado para R195.62 não houve diferença entre explícito ou implícito para o driver OpenCL. A NVIDIA melhorou o gerenciamento do work group size, funções nativas e performance.

OpenCL NDRange Configuration



GeForce GTS 250 + R195.39:

- 800×800 implicit work group size: 12 FPS
- 800×800 implicit work group size + native_func = 19 FPS
- 800×800 explicit work group size: 31 FPS
- 800×800 explicit work group size + native_func = 32 FPS

Alguns resultados obtidos:

Programa: GPU Caps 1.8.2 PRO [10]

Sistema Operacional: Windows Vista SP2 32-bit

Memória: 2 GB 1333 DDR3

CPU: Intel Core 2 Extreme CPU X9650 @ 3.00 GHz

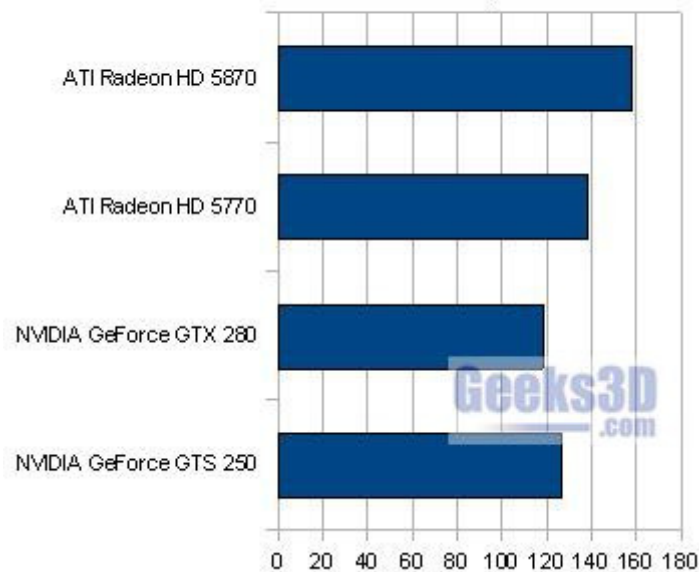
NVIDIA Driver: R195.62 [8]

AMD Driver: Catalyst 9.12 [11]

A malha foi inicializada com 511x511 segmentos que equivalem a 512x512 vértices. Esta malha teve 522,242 faces e 262,144 vértices. A demonstração mostra uma superfície deformada por uma espécie de onda senoidal. O código OpenCL faz todo o trabalho pesado, calculando as posições dos vértices na deformação de superfície e calcula os vértices normais para uma boa luminosidade. Uma vez que todos os cálculos são finalizados, os dados são enviados para OpenGL VBOs (Vertex Buffer Object). A renderização é feita com OpenGL 3 context. O eixo X representa a média de FPS (Frames Por Segundo).

CPU Caps Viewer OpenCL Test

Surface Deformer 512x512 All Optimizations



Analisando o gráfico, notamos que as placas Radeon tiveram o melhor desempenho, mas um estranho resultado pode ser observado, onde a GeForce GTS 250 foi melhor que a GTX 280. Como a arquitetura de ambas as placas é diferente, uma hipótese é que a série GT 200 não foi explorada o suficiente pelo driver da NVIDIA.

Outro resultado é que a placa HD5870 não foi duas vezes mais rápida que a HD5770, apesar de ter o dobro de performance em nível de hardware. Neste teste, em particular, essa pequena diferença pode ser explicada pela interferência do driver OpenCL ou por sobrecarga. O código OpenCL não é muito complexo para manter a GPU ocupada durante diversos ciclos. Em suma, as chamadas OpenCL são pesadas e minimizam a diferença de performance entre ambas GPUs. Para notar uma importante diferença é necessário aumentar a carga do kernel com o seguinte parâmetro:

```
GpuCapsViewer.exe /cl_kernel_workload_sin=10000
```

Este parâmetro (que por padrão é zero) permite aumentar a carga do kernel adicionando um loop feito sobre alguns calculos de seno, cosseno e raízes, sem afetar a deformação da superfície. Este parâmetro foi inserido apenas para consumir recursos. Para as Radeons nota-se o mínimo de interferência do driver OpenCL no resultado e então é possível notar a diferença (o dobro) de performance entre a HD5870 e a HD5770. Atenção! 10000 é um número muito elevado para GeForce!

Os testes feitos pela PC Impact [12] trazem mais resultados do mesmo teste feito no Windows 7.

	Interop	Native Sin	Explicit	4670	4890	5750	5870
Mesh Deformer - 128x128	√	√	√	507	460	730	906
Mesh Deformer - 256x256				202	163	234	308
Mesh Deformer - 512x512				52	43	82	95
Mesh Deformer - 1024x1024				13	10	22	27
Mesh Deformer - 100	√	√	√	486	441	456	628
Mesh Deformer - 500				338	388	414	537
Mesh Deformer - 1000				278	343	352	444
Mesh Deformer - 5000				112	170	154	251
Mesh Deformer - 100	√	√	X	-	469	468	583
Mesh Deformer - 500				-	433	409	510
Mesh Deformer - 1000				-	395	353	446
Mesh Deformer - 5000				-	221	159	242
4D Quaternion Julia Set	X	√	√	2	39	35	65
			X	1	30	26	52
Particles	X	√	√	24	17	27	34
		X		24	17	26	34
		√		24	17	26	34
		X		24	17	27	34
		√	X	24	19	35	40
		X		24	19	35	40
		√		24	19	34	40
		X		24	19	35	40
PostFX - Global Mem	√	√	√	-	20	19	35
			X	-	17	16	32
PostFX - Local Mem	√	√	√	-	-	-	-
Moyenne				121,3	157,8	173,9	227,4

OpenCL Radeon Performance

	Interop	Native Sin	Explicit	ION	GT 220	9600GT	GTS 250	GTX 275
Mesh Deformer - 128x128				77	329	467	591	442
Mesh Deformer - 256x256	√	√	√	49	194	278	356	286
Mesh Deformer - 512x512				24	87	120	128	117
Mesh Deformer - 1024x1024				7	31	33	35	34
Mesh Deformer - 100				85	320	446	563	429
Mesh Deformer - 500	√	√	√	77	295	413	540	411
Mesh Deformer - 1000				69	264	377	516	380
Mesh Deformer - 5000				40	146	222	362	274
Mesh Deformer - 100				85	314	447	589	432
Mesh Deformer - 500	√	√	X	77	293	412	544	417
Mesh Deformer - 1000				69	263	376	517	407
Mesh Deformer - 5000				40	149	222	363	317
4D Quaternion Julia Set				√	√	√	7	30
	X	X	4	15		11	22	50
	X	√	7	29		33	54	55
Particles	√	√	√	14	76	121	118	167
		X		14	75	121	118	166
		√		14	75	121	118	166
		X		14	76	121	118	167
	√	√	X	14	75	106	104	165
		X		14	75	106	104	164
		√		14	75	106	103	165
		X		14	75	105	104	165
	X	√	√	10	55	76	73	79
		X		10	55	76	73	78
		√		10	55	76	73	78
		X		10	55	76	73	79
PostFX - Global Mem	√	√	√	4	17	13	16	60
X			4	17	13	15	59	
PostFX - Local Mem	√	√	√	5	18	26	52	63
Moyenne				29,4	121,1	171,8	216,7	197,6

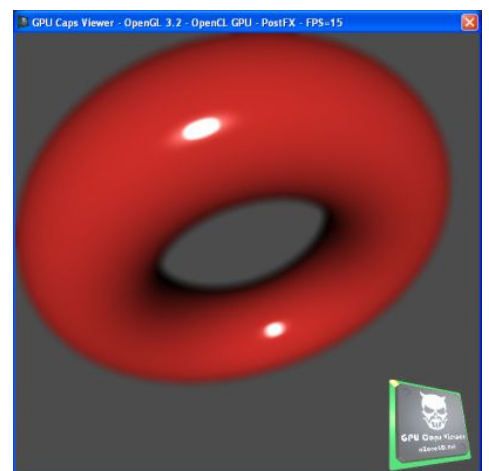
OpenCL NVIDIA Performance

Segundo Teste: PostFX

Esta demonstração é uma implementação direta do NVIDIA oclPostprocessGL disponível no NV's GPU Computing SDK [13]. O PostFX aplica blur na imagem final. A novidade nesta demonstração é o uso da memória local e global. O programa GPU Caps possui uma opção em linha de comando para ativar o uso de memória local (/cl_demo_use_local_mem). Por padrão, o uso de memória local é desativado por causar falhas nas placas Radeon.

A memória local possui acesso muito rápido aos chips de memória (scratchpad memory) e é muito mais rápida que a memória global (memória gráfica, localizada fora da GPU). Na implementação original da NVIDIA é usada apenas memória local com explicit workgroup size. Foram adicionadas opções para ativar ou desativar o uso de memória local, assim como o explicit workgroup size.

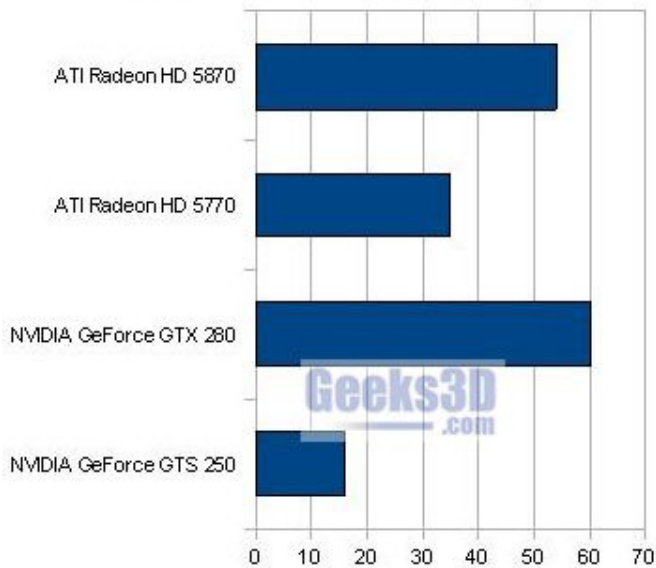
Todos os testes foram realizados no mesmo computador, cuja configuração foi informada anteriormente.



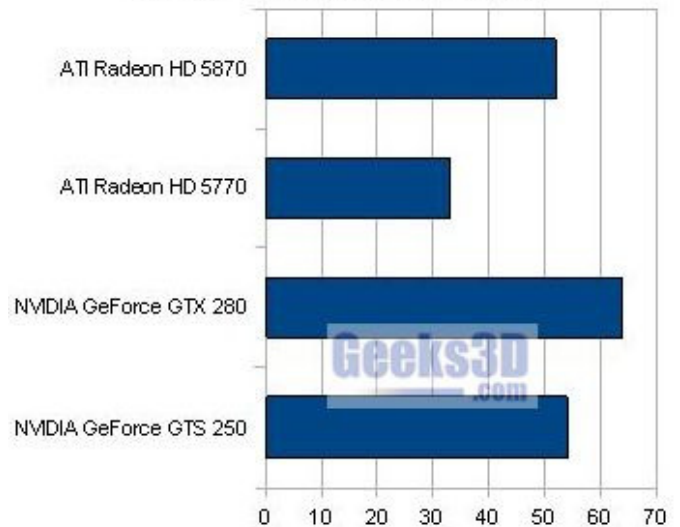
Demonstração do OpenCL Post FX

Aqui estão alguns resultados do pós processamento de uma janela de 600x600:

GPU Caps Viewer OpenCL Test
PostFX - 600x600 - Global Memory



GPU Caps Viewer OpenCL Test
PostFX - 600x600 - Local Memory

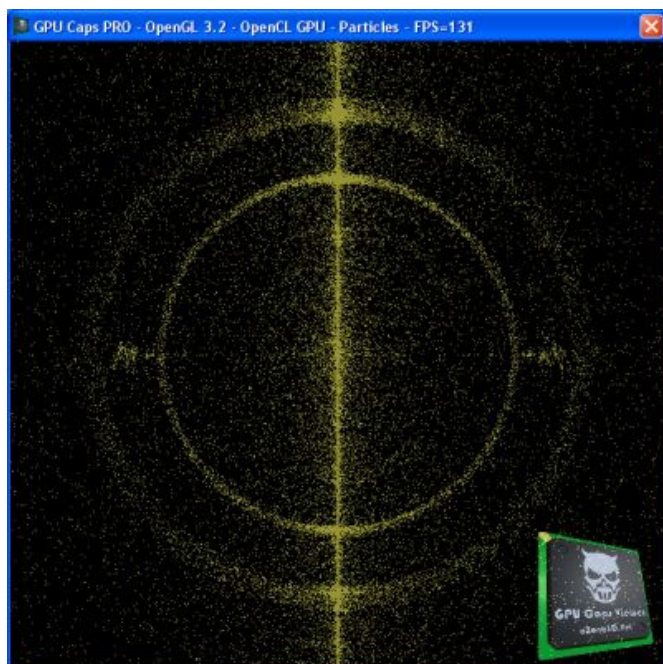


Para o teste realizado, as placas NVIDIA utilizando os primeiros drivers OpenCL apresentaram grande ganho de performance quando usado explicit work group size. Agora esta declaração não é mais verdadeira. Estes gráficos mostram um fato interessante: a memória local tem grande impacto na GeForce GTS 250 enquanto representa pouco para a GTX 280. Em contrapartida notamos que não faz diferença alguma para as Radeons.

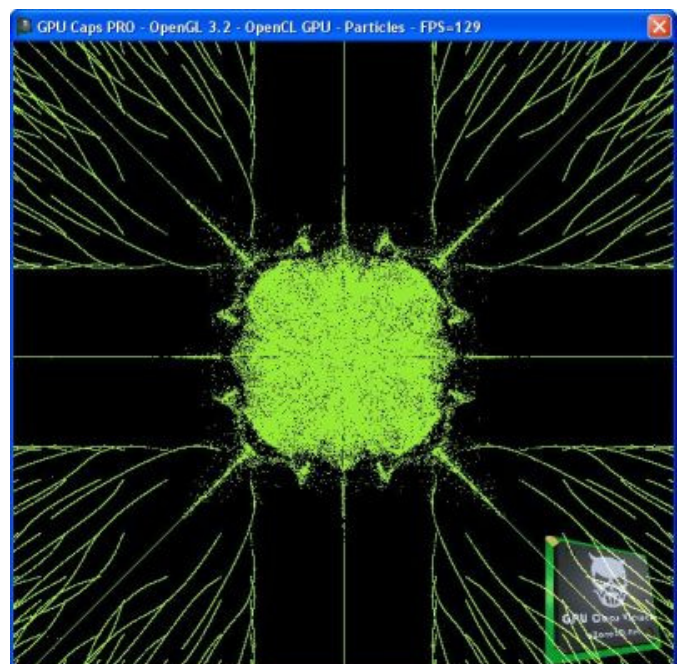
Este kernel de pós processamento é bastante exigente e podemos notar a diferença entre a HD 5870 e a HD 5770. Também podemos notar que a GTX 280 dominou o teste. O código vem com o NVIDIA OpenCL SDK e certamente está otimizado para o hardware da NVIDIA.

ATENÇÃO! Para os usuários de Windows XP ou Seven, o uso de memória local em placas Radeon causa falha no sistema e o VPU Recover redefine o driver gráfico (faz reset do driver). Isso não acontece no Windows Vista.

Terceiro Teste: 1.000.000 de partículas



Demonstração do OpenCL com 1,000,000 partículas



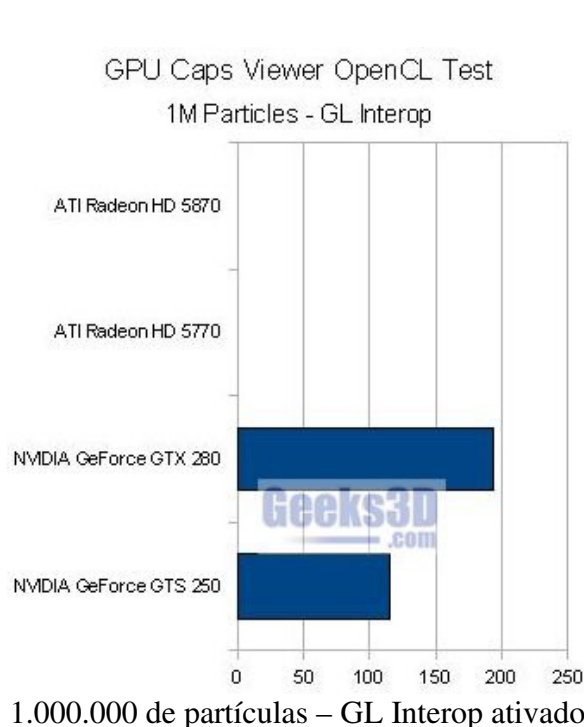
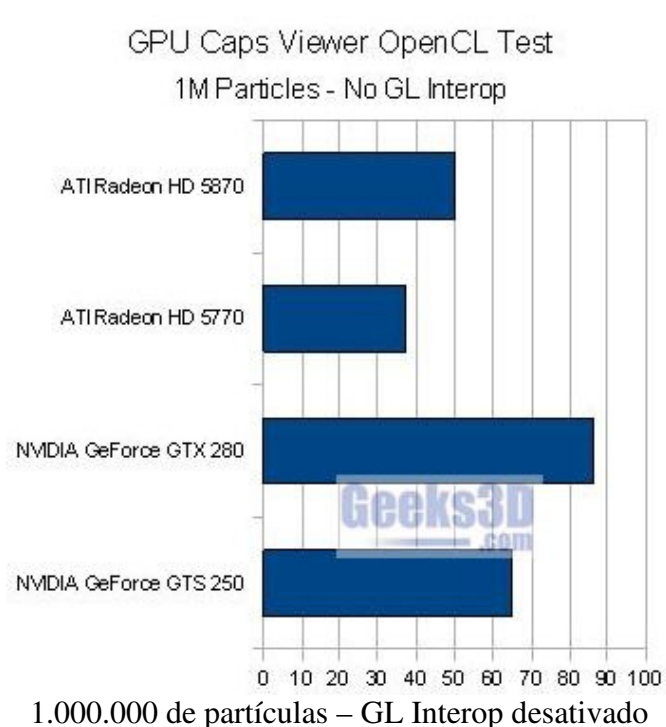
Demonstração com 1,000,000 partículas (com seno)

Esta demonstração é uma implementação direta da com 1.000.000 de partículas [14]. Existe um arquivo de lote (Start_OpenCL_Particles.bat) na pasta GPU Caps que permite o uso da versão normal ou da versão com seno. É possível mudar o número de partículas (/cl_particles_gpu=2000000 por exemplo).

Este programa usa uma característica interessante do OpenCL: interoperabilidade GL (GL Interop), que é a habilidade de comunicação entre OpenCL e OpenGL. O OpenCL pode atualizar diretamente os buffers do OpenGL sem passar pela aplicação do host. Por padrão, GL Interop está desativado no GPU Caps. Para ativa-lo, use o parâmetro na linha de comando: /cl_demo_gl_interop_enabled.

Quando o GL Interop está ativado, a posição das partículas é calculada pelo kernel OpenCL e o OpenCL automaticamente atualiza o OpenGL VBO com as novas posições. O OpenGL VBO é usado para renderização das partículas.

Quando o GL Interop está desativado, a posição das partículas ainda são calculadas pelo kernel do OpenCL mas agora todas as posições são copiadas (primeira cópia) do OpenCL para o buffer de memória do host. Esse buffer é então usado para atualizar (segunda cópia) o OpenGL VBO para renderização de partículas. O GL Interop evita essas cópias extras.

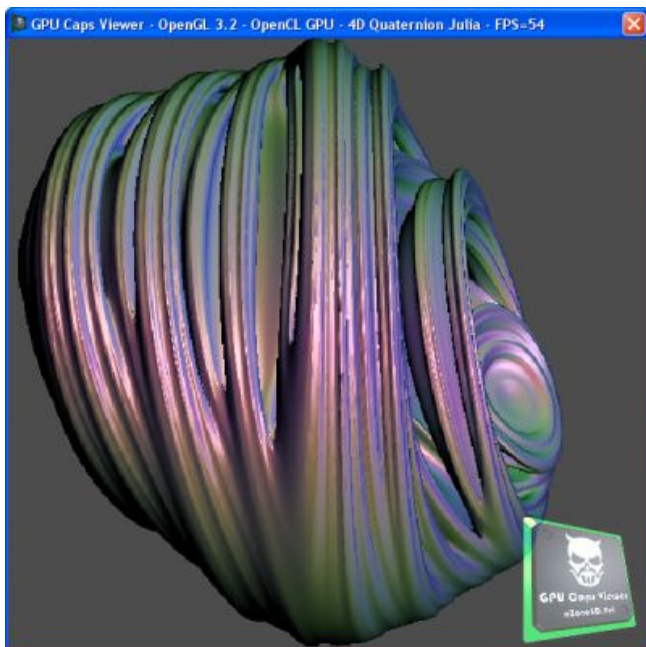


Atualmente as placas Radeon não suportam GL Interop devido a falta da extensão **cl_khr_gl_sharing** na plataforma AMD (é por esse motivo que a função GL Interop está ativada por padrão no programa GPU Caps) Para mais detalhes, leia o artigo [15].

Na plataforma NVIDIA, essa extensão é suportada e melhora a performance de maneira impressionante onde temos o resultado de 85 FPS para 200 FPS na GTX 280. A melhora também é notada na GTS 250, deixando as NVIDIA bem a frente das Radeons. A dúvida é: há falta de otimização do OpenCL ou é uma falha na implementação da AMD?

Quarto Teste: 4D Quaternion Julia

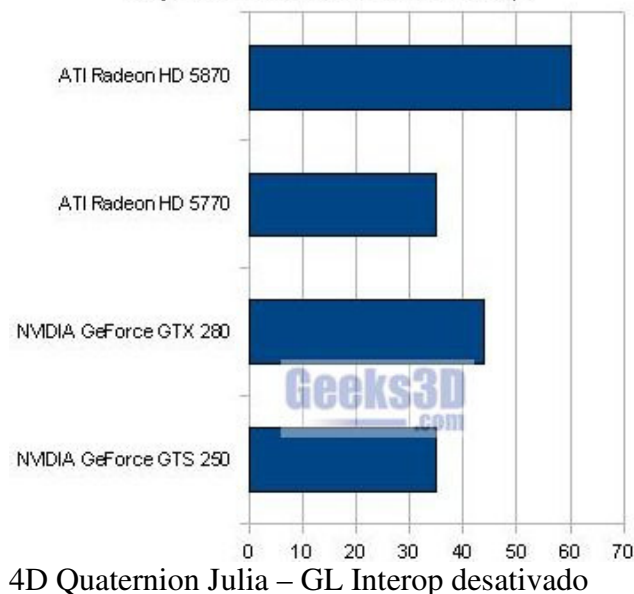
Este exemplo é uma importação direta do código Ray Traced Quaternion Julia Set sample [16]. Este código OpenCL está disponível na pasta media do GPU Caps. Este código possui a mesma opção de GL Interop que o teste com 1.000.000 de partículas. Infelizmente a imagem do resultado do teste com GL Interop ativado foi perdida e o autor da versão original pede desculpas pelo descuido.



Demonstração do OpenCL 4D Quaternion Julia

GPU Caps Viewer OpenCL Test

RayTraced Julia 4D - No GL Interop



Conclusão

Estes resultados refletem o atual estado dos drivers OpenCL da AMD e NVIDIA. Os resultados podem variar de acordo com a versão dos drivers Catalyst e ForceWare (além dos ajustes feitos no código OpenCL).

Com os primeiros drivers (R190.89 ou R195.39) as placas GeForce precisaram de otimização no código OpenCL para demonstrar o potencial que possuem enquanto as placas Radeon não foram muito afetadas por otimizações (especialmente explicit work group size e funções nativas). Agora com a última versão R195.62, a NVIDIA melhorou a implementação do OpenCL e ficou muito melhor.

Ainda existem alguns problemas como a GTS 250 estar mais rápida que a GTX280 em alguns testes ou a falta de suporte GL Interop na plataforma AMD.

Atualmente o OpenCL não está maduro e é totalmente compreensível: a primeira implementação de OpenCL surgiu apenas há alguns meses atrás.

Novas avaliações serão feitas quando os drivers OpenCL da AMD estiverem mais estáveis e com suporte a GL Interop (sem a necessidade de instala o ATI Stream SDK) e logo a NVIDIA disponibilize a GT100 [17].

Glossário

ALU (Arithmetic Logic Unit) Unidade Lógica Aritmética é um circuito que realiza operações lógicas e cálculos no computador.

API (Application Programming Interface, ou seja, Interface de Programação de Aplicativos) é um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por programas aplicativos que não querem envolver-se em detalhes da implementação do software, mas apenas usar seus serviços.

SIMD (Single Instruction, Multiple Data) Uma instrução, Múltiplos dados descreve um método de operação de computadores com várias unidades operacionais em computação paralela. Neste modo, a mesma instrução é aplicada simultaneamente a diversos dados para produzir mais resultados. O modelo SIMD é adequado para o tratamento, cuja estrutura é muito regular, como as matrizes e vetores. Esse tipo de máquina opera aplicando uma única instrução a um conjunto de elementos de um vetor. Sendo uma máquina que aplique a n elementos uma determinada instrução e o vetor t contenha os elementos a serem processados, t terá todos seus elementos calculados n vezes mais rápido que uma máquina SISD na mesma tarefa.

SISD (Single Instruction, Single Data) Uma instrução, Um dado, ou seja, fluxo único de instruções e também de dados. É um dos tipos de arquitetura mais simples, já que opera apenas um dado a cada instrução. Processadores que implementam esse modelo só aplicam uma instrução por ciclo nos dados de entrada, sendo de baixo poder de cálculo.

Referências

[1] GPU Caps Viewer 1.8.2 (disponível em: <http://www.geeks3d.com/20091228/gpu-caps-viewer-1-8-2-available/> ou no site oficial: http://www.ozone3d.net/gpu_caps_viewer/).

[2] OpenCL Overview (<http://www.khronos.org/opencv/>).

[3] Explicações de Neil Trevett <http://www.geeks3d.com/20091106/opencv-and-gpu-computing-latest-news/>

[4] Radeon HD 5760
<http://www.geeks3d.com/20100114/ati-radeon-hd-5670-direct3d-11-and-opengl-3-2-for-the-masses/>

[5] OpenCL para Radeon HD 5870 com Catalyst 9.12 pode ser encontradas neste endereço:
<http://www.geeks3d.com/20091221/ati-catalyst-9-12-opengl-and-opencv-details/>

[6] Mais informações sobre o pacote NVIDIA OpenCL SDK:
<http://www.geeks3d.com/20091009/nvidia-release-the-gpu-computing-sdk-with-opencv-support-and-demos/>

[7] ForceWare 195.39
<http://www.geeks3d.com/20091029/forceware-195-39-nvidias-first-graphics-drivers-with-public-opencv-support/>

[8] NVIDIA WHQL R195.62
<http://www.geeks3d.com/20091126/nvidia-forceware-195-62-whql/>

[9] NDRange Configuration
<http://www.ks.uiuc.edu/Research/gpu/files/opencvintrowebinar.pdf>

[10] GPU Caps 1.8.2 PRO
http://ozone3d.net/gpu_caps_viewer/gpu_caps_pro.php

[11] AMD Catalyst 9.12 hotfix
<http://www.geeks3d.com/20091218/ati-catalyst-9-12-hotfix-with-opencv-support/>

[12] PC Inpact
<http://www.pcinpact.com/>

[13] NV's GPU Computing SDK
<http://www.geeks3d.com/20091009/nvidia-release-the-gpu-computing-sdk-with-opencv-support-and-demos/>

[14] 1.000.000 Particles Demo
<http://www.geeks3d.com/20091106/opencv-and-gpu-computing-latest-news/>

[15] Atualização do ATI Stream SDK v2.0.0
<http://www.geeks3d.com/20091222/gpu-caps-viewer-1-8-1-updated-for-ati-stream-sdk-v2-0-0/>

[16] Ray Traced Quaternion Julia Set sample
http://developer.apple.com/mac/library/samplecode/OpenCL_RayTraced_Quaternion_Julia_Set_Example/index.html

[17] NVIDIA GT100
<http://www.geeks3d.com/20100118/nvidia-gf100-architecture-details/>