# GPUs - Graphic Processing Units

Bruno P S Rocha
Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
bpontes@dcc.ufmg.br

Rone I Silva
Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
rone@dcc.ufmg.br

## Abstract

*Neste trabalho fazemos um estudo aprofundado sobre a GPUs - Graphic Processing Units. Estudamos o seu conceito e sua história, e apresentamos pontos arquiteturais, como unidades funcionais e o funcionamento básico de seu pipeline, evidenciando funcionalidades providas pelo hardware. Apresentamos um exemplo de uma GPU atual e fazemos também um estudo sobre desempenho. Por fim, mostramos como as GPUs são programadas e apontamos tendências para o futuro.*

## 1 Introdução

Em aplicações gráficas, as funções de *transform and lightining* (*T&L*) utilizam muito recurso computacional, por isso requerem um tratamento diferenciado pelo *hardware*. *Transforms* são operações que, à partir de objetos descritos por coordenadas em três dimensões (3D), calculam como será exibida a cena em duas dimensões (2D). *Lightining* são operações que calculam o brilho dos *pixels* das superfícies dos objetos, levando em consideração a inclinação de tais superfícies, a quantidade de luz e a disposição da luz em cada cena. Tais operações podem ser processadas pela unidade de ponto flutuante da CPU (FPU - *Floating Point Unit*). Contudo, quando o bom desempenho do processamento gráfico é um fator essencial, como por exemplo em jogos, tais operações são transferidas para unidades especializadas, as GPUs.

GPU (*Graphics Processing Unit*, ou Unidade de Processamento Gráfico) é, em linhas gerais, um microprocessador destinado a realização de operações de *T&L*. Tais processadores são montados em placas conectadas ao barramento da placa mãe dos computadores para realizar o processamento das operações gráficas e enviar os dados processados para um monitor [22]. Com isso, além do processamento gráfico ficar mais eficiente, a carga de trabalho da CPU reduz, melhorando o desempenho da máquina.

Os primeiros computadores que implementaram conceitos de GPU em um *hardware* separado da CPU foram o *Atari 800* [20] e o *Atari 5200* [19], no final da década de setenta. Eles possuíam um co-processador, denominado ANTIC (*Alpha-Numeric Television Interface Circuit*), que através de acesso direto á memória (DMA - *Direct Memory Access*) enviava sinais para a televisão ou monitor conectado ao micro computador. Nos anos oitenta, o *Commodore Amiga* [18] foi o primeiro microcomputador a incorporar um *blitter* em seu *hardware* de vídeo. *Blittler* é um programa capaz de, à partir de dois padrões bitmap, obter um terceiro padrão, que será exibido. Essa operação é muito útil quando tem-se um fundo (*background*) sob a cena principal. Tal característica fez com que o *Commodore Amiga* diminuísse o trabalho realizado pela CPU, melhorando seu desempenho. Pouco tempo depois, o sistema gráfico *IBM 8514* [17] foi o primeiro a implementar diretivas 2D em *hardware*. Antes dele, outros computadores (como alguns *workstations*) já possuíam essa característica, mas o preço de tais máquinas era muito alto.

No início dos anos noventa foi criado o *S3 86C911*, o primeiro *single-chip* dedicado a processamento de gráficos 2D em PC's. A partir de então, várias outras placas foram criadas e lançadas no mercado. Em relação ao processamento 3D, inicialmente alguns fabricantes incorporaram à suas placas 2D funções destinadas ao processamento 3D. Contudo, o desempenho não alcançou um patamar esperado e tais projetos não evoluíram de forma considerável. Somente em 1995 foi lançada o primeiro *hardware* dedicado exclusivamente ao processamento de gráficos 3D, a GeForce 256 [14] e com ela o conceito de GPU. A figura 1 mostra um gráfico de evolução e projeção do desempenho de GPUs, segunda a nVidia.

Este documento está organizado como se segue. Na seção 2 apresentamos trabalhos relacionados com es-


**Figura 1. Evolução das GPUs**

tudos em GPUs, focados ou gerais. Na seções 3 e 4 definimos o conceito básico de GPU e apresentamos sua arquitetura, respectivamente. Na seção 5 mostramos um exemplo de GPU atual, com base nos pontos arquiteturais da seção que a precede. Uma análise de desempenho, evidenciando quais aspectos possuem maior impacto na performance, é mostrada na seção 6. A seção 7 mostra como são programadas as GPUs. Finalmente, a seção 8 aponta tendências para o futuro e na seção 9 colocamos nossas conclusões.

## 2 Trabalhos Relacionados

As GPUs possuem arquiteturas fechadas. Com isso, muitos dos trabalhos relacionados tentam deduzir aspectos arquiteturais através de funcionalidades conhecidas e informações publicadas por fabricantes. Esse tipo de estudo pode ser visto em [22, 1, 6, 16, 13, 9]. Uma análise de desempenho de GPUs pode ser encontrada em [7].

Os fabricantes também divulgam aspectos de suas GPUs, para facilitar o trabalho de desenvolvedores. A nVidia, por exemplo, divulga guias de programação de GPUs [15, 5], tutoriais sobre tecnologias de *shaders* [4, 2, 3, 8, 12], e também documentos que explicam estruturas e conceitos gerais de arquiteturas específicas, como a GeForce256 [14], a GeForce3 [11], e a GeForce Series 6 [10].

## 3 Graphic Processing Units

Uma GPU é um microprocessador dedicado exclusivamente a processar gráficos, em sua maioria em 3 dimensões. As GPUs geralmente funcionam em placas conectadas ao computador através de *slots* de expansão. A comunicação entre CPU e GPU ocorre através de um barramento de expansão da placa-mãe do computador. Existem GPUs, principalmente as mais antigas, que também utilizavam o barramento para fazer acesso à memória principal do computador. Atualmente as GPUs utilizam memória contida na sua própria placa, através de uma interface de alta velocidade.

A figura 2 mostra em destaque as principais partes de uma placa gráfica. As "saídas" são os conectores onde os cabos dos monitores são conectados. Existem vários tipos de conectores, os mais encontrados são: VGA, S-VHS, HDMI, RCA e DVI. A "interface" faz a ligação da GPU com o barramento da placa-mãe, através de quatro tipos *slots* de expansão: ISA, PCI, AGP e PCI Express. As memórias podem ser visualizadas como os retângulos negros estrategicamente fixados ao redor do processador. Na figura, a GPU é indicada mas não pode ser vista, pois encontra-se abaixo do *cooler*. Muitas de suas características de fabricação são mantidas em segredo pelos seus fabricantes para evitar cópias.

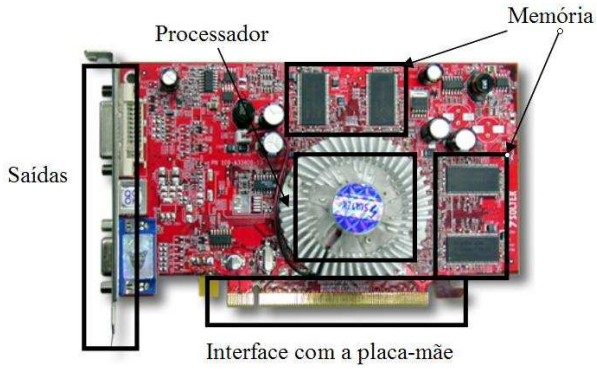


**Figura 2. Estrutura de uma placa gráfica**

## 4 Arquitetura

Por se tratar de uma área desenvolvida e pesquisada principalmente no meio privado, as GPUs possuem arquiteturas fechadas, ou seja, suas especiicações detalhadas são mantidas em sigilo pelos fabricantes. No entanto, por serem programáveis com conjuntos de instruções abertos e serem utilizadas extensivamente, muitas de suas características arquiteturais principais são conhecidas por desenvolvedores.

Podemos a princípio citar alguns aspectos gerais comuns à grande maioria das GPUs. Sabemos que são processadores de arquitetura SIMD (*single instruction, multiple data*) paralelizada. Em sua arquitetura, existem múltiplas unidades de processamento paralelo de-

notadas "pipelines de pixel" [1]. Cada "pipeline" compreende um conjunto de unidades funcionais que processam vértices e pixels. Sabemos também que as GPUs, em sua maioria, não podem ser consideradas processadores vetoriais genéricos. Isso acontece principalmente pela forma como acessam a memória, não podendo ler e escrever em áreas iguais.

Uma GPU padrão trabalha com 5 tipos básicos de estruturas de memória [1]. Existem os objetos "bufferizados", gerados pela CPU e armazenados na memória de vídeo, com baixa modificabilidade, como os dados de vertex de um dado material. No processador, existem os registradores uniformes, que garantem o fluxo de dados constante pelo *pipeline* e são formados de registradores de propósito geral e de registradores específicos de tabelas de estado (matrizes de projeção/modelo, luzes, etc.). Existem ainda os registradores interpolados, que servem para trabalhar com dados por vértices individuais de um polígono. Ainda no processador, temos os registradores temporários, para usos diversos e normalmente usados para cálculos dos *shaders*. Finalmente, temos as texturas, que ocupam a maior parte da memória de vídeo e são as estruturas com acesso mais caro.

### 4.1 Shaders

Antes de prosseguir no estudo da arquitetura das GPUs, é importante mencionar os *shaders*. Um *shader* consiste em um pequeno programa que roda na GPU em estágios específicos do *pipeline* de renderização [1].

Os *shaders* foram introduzidos em 2001 com as GPUs nVidia GeForce3 e ATI Radeon 8500, primeiramente com o *vertex shader*. Esse tipo de *shader* trabalha sobre vértices no espaço 3D, deformando ou transformando elementos. Utilizados para criar efeitos de lente, embassamento de imagens, imagens submarinas, amassamento e entortamento de objetos, movimentos de tecidos, cabelo e água, entre outros.

Nos anos de 2002 e 2003 foi introduzido o *pixel shader*, com as séries nVidia GeForce FX e ATI Radeon 9600 a 9800. Estes são usados para aplicar operações sobre pixels individuais, afetando sua cor final. São usados em efeitos gráficos complicados, como os de iluminação e sombra [6, 9]. Às vezes, são também conhecidos como processadores de fragmento.

[1]Note que o termo *pipeline* utilizado aqui não tem o exato significado do estudo de arquitetura de computadores. Neste documento, fazemos referência a 2 tipos de *pipeline*: o de pixel, citado acima, que corresponde a cada uma das unidades de processamento de pixel, e o *pipeline* de renderização, que é o *pipeline* principal da GPU vista como um todo, englobando os passos de renderização.

Os *shaders* são hoje um ponto fundamental no desenvolvimento de aplicações de gráficos intensos, como jogos. Através da definição de micro-programas que afetam determinados vértices e pixels e são executados diretamente na GPU, desenvolvedores conseguem atingir níveis de qualidade gráfica antes impossíveis. Os próprios fabricantes dão suporte total ao seu uso, provendo tutoriais [4, 2, 3, 8, 12] e linguagens de alto nível para programação dos mesmos (veja seção 7).

### 4.2 Unidades Funcionais

Um processador gráfico contém diversas unidades funcionais diferentes. Algumas delas podem ser identificadas para nos dar uma idéia geral do quão poderoso é um dado processador gráfico em uma dada tarefa. Citamos abaixo as principais unidades de uma GPU [6].

**Pixel Processors (Pixel Shader Units):** Esta é a unidade dedicada exclusivamente aos *pixel shaders*. Todos seus cálculos são feitos diretamente sobre valores de pixels individuais. Como pixels representam valores de cores, essas estruturas são responsáveis por vários tipos de efeitos visuais. O número de *pixel shaders* em uma placa gráfica é normalmente a principal métrica de desempenho.

**Vertex Processors (Vertex Shader Units):** Similar às unidades de *pixel shader*, processadores de vértices são componentes da GPU projetados para processarem *shaders* que afetam apenas vértices. Como mais vértices indicam objetos 3D mais complexos, *vertex shaders* são importantes em cenas com muitos e complexos objetos 3D. Eles são, no entanto, claramente menos relevantes para a performance geral da GPU do que os *pixel shaders*.

**Texture Mapping Units (TMUs):** Texturas precisam ser endereçadas e filtradas. Esse trabalho é feito pelas TMUs, que trabalham em conjunção com as unidades de *shaders* de pixels e vértices. Esta unidade é a responsável por aplicar operações de texturas a pixels.

**Raster Operator Units (ROPs):** Os processadores de operações de rasterização são responsáveis por escrever dados de pixels na memória. A velocidade em que isto é feito é conhecida como *fill rate*. ROPs e *fill rates* costumavam ser uma métrica muito importante nos primórdios das placas gráficas 3D. Apesar de seu trabalho ser importante, ele não é mais o gargalo computacional que já foi um dia, e não é mais usado como indicador de performance de GPUs.

**Pixel Pipelines:** No contexto das GPUs, o termo *pixel pipeline* não tem uma definição formal. No entanto, na maioria das GPUs ele é usado para descrever a estrutura composta por um processador de pixel ane-

xado a uma TMU dedicada. Desta forma, o número de *pixel pipelines* de uma GPU representa o número de operações sobre pixels que ela trata paralelamente, ou seja, o número de pixels processados por pulso de *clock*.

## 4.3 O Pipeline de Renderização Gráfica

O processamento de imagens na GPU segue uma seqüência básica denominada *pipeline* de renderização gráfica, que não deve ser confundido com as unidades de *pixel pipeline*. O *pipeline* completo, por definição, inclui também passos executados ainda na CPU. A figura 3 mostra um diagrama básico de como é estruturado o *pipeline*.

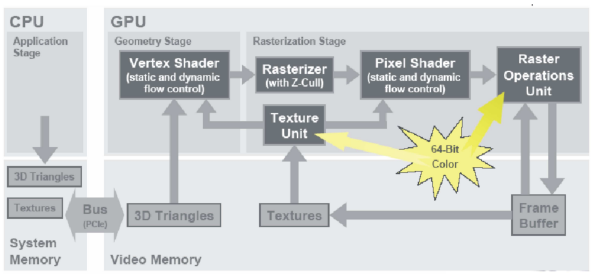


**Figura 3. Pipeline de renderização gráfica [9]**

O processamento de gráficos começa no lado da CPU, no estágio da aplicação. Nesse estágio, código de aplicações gráficas é escrito sobre APIs gráficas como DirectX e OpenGL, que por sua vez acessam os *drivers* das placas gráficas. Os *drivers* são peças de *software* fechadas, cujas implementações são segredos comerciais, e são hoje um dos principais gargalos em muitos programas de GPU.

O *driver* é o responsável pelo acesso direto à GPU. Ele traduz chamadas em alto nível para instruções de GPU e executa procedimentos para que instruções e dados sejam enviados pelo barramento gráfico. A partir desse ponto, o processamento na GPU em si começa.

Dentro da GPU, o processamento é feito em uma seqüência bem definida. O primeiro passo é a execução dos *vertex shaders*, que fazem transformações em polígonos no espaço 3D, trabalhando com polígonos na memória da GPU. Como já explicitado, os *shaders* são programáveis e utilizam código enviado pela aplicação, através do *driver*.

Depois disso, entra-se na fase de rasterização. A unidade de rasterização faz operações por pixel, para determinar cores de pixels no *frame buffer*. Para isso, converte primitivas 3D em fragmentos, que são estruturas de dados que encapsulam possíveis pixels a serem exibidos na tela. Então, faz operações de interpolação com texturas e iluminação, combinações de cores, testes de pixel como testes alpha e estêncil, determina áreas de visibilidade e faz a composição do *frame buffer*.

No último estágio do processamento dos fragmentos, são acionados os *pixel shaders*. Programáveis, eles acessam instruções vindas do *driver* e executam operações sobre cada pixel, determinando suas cores finais. Finalmente, são executadas *raster operations*, que trabalham já no *frame buffer*, checando questões de profundidade (*Z-buffer*) e combinando cores já enviadas para ele (*blending*) [9, 1]. Uma visão geral dos passos pode ser vista na figura 4.

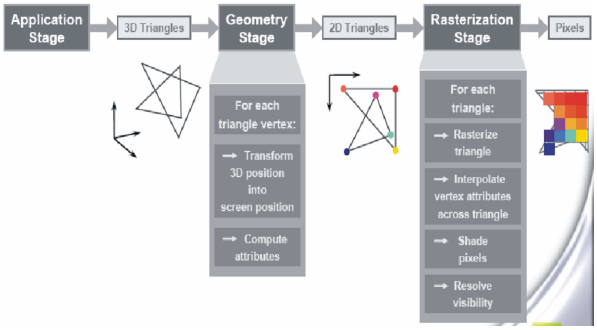


**Figura 4. Passos conceituais executados no pipeline de renderização gráfica [9]**

## 4.4 Sistema de Memória

Apesar da arquitetura interna da GPU ser o ponto principal do nosso estudo, a memória é de fundamental importância para um bom desempenho do sistema. Sistemas de memória lentos ou pequenos se tornam gargalos em sistemas que possuem bons processadores.

Os tipos de memória utilizados em GPUs são de duas principais categorias: SDR (*single data rate*) e DDR (*double data rate*), que fazem a transferência de dados de forma síncrona. Atualmente, SDR é considerado obsoleto, uma vez que as memórias do tipo DDR trabalham com duas vezes mais dados por ciclo de *clock*. As memórias dos tipo DDR2 e DDR3 possuem as mesmas características das memórias DDR, diferenciando somente na forma de fabricação para aceitar *clocks* mais elevados. Importante mencionar que o *clock* da GPU e o *clock* da memória não possuem correlação entre si, tendo valores distintos.

O tamanho da memória varia entre os diversos modelos de placas gráficas encontrados no mercado. Podem ser encontradas placas com memória de 128 MB, 256 MB, 512 MB ou mais, nos modelos mais avançados.

O tamanho da memória é importante, principalmente, para o armazenamento de texturas. Tais estruturas ocupam muito espaço na memória e por isso, quando armazenadas na memória da GPU, evitam a busca de dados na memória principal do computador, operação que pode se tornar o gargalo do sistema.

No entanto, todo o sistema de memória é dependente de um bom sistema de barramento que a interligue com o processador gráfico. Podem ser encontrados modelos de GPUs com barramento de largura variando entre 64, 128 e 256 bits. Logicamente, quanto mais largo o barramento maior quantidade de dados que poderá trafegar de uma única vez, diminuindo o tempo de acesso.

### 4.5 Barramentos de Dados

Para se comunicar com a CPU, é importante que a GPU tenha acesso a um barramento de dados rápido. O rápido crescimento no uso das GPUs estimulou o desenvolvimento de barramentos específicos para suas necessidades. Hoje em dia existem três tipos de barramentos em uso: PCI, AGP e PCI Express.

O padrão PCI (*Peripheral Components Interconnect*) não foi desenvolvido especificamente para GPUs e hoje é considerado ultrapassado. O AGP (*Accelerated Graphics Port*) foi feito para remover o gargalo dos barramentos, com largura de banda de até 2,16 GB/s (AGP 8x). Hoje em dia, o padrão está sendo substituído pelo PCI Express x16, que é um canal *dual-channel* com largura de banda de 4 GB/s. No entanto, o padrão AGP 8x fornece largura de banda mais que necessária para a maioria das GPUs atuais, já fazendo com que o barramento deixe de ser um limitador de desempenho. O estudo aprofundado de barramentos não está no escopo deste documento.

### 4.6 Múltiplas GPUs

O uso de múltiplas GPUs em paralelo não é uma idéia nova. Nos primóridios dos gráficos 3D, a empresa 3dfx foi pioneira em tecnologia de múltiplas GPUs. No entanto, a empresa foi a falência pouco antes de disponibilizar a tecnologia a consumidores. A tecnologia foi reintroduzida recentemente, com a nVidia SLI e a ATI Crossfire.

Apesar do uso de GPUs em paralelo aumentar em muito o desempenho do sistema, a escolha para compra e instalação de múltiplas placas gráficas não é simples. Primeiramente, o computador a receber mais de uma placa gráfica deve ter considerações de energia, necessitando de fontes mais potentes que o normal, e calor, com cuidados especiais de resfriamento. Além disso, configurações multi-GPU necessitam de placas mãe com *chipsets* que as suportam, geralmente mais caros. As próprias placas gráficas que suportam paralelismo são geralmente os modelos mais caros do mercado.

É importante observar também que o ganho de desempenho não é linear com o número de GPUs instaladas. Nos modelos atuais, duas GPUs em paralelo têm desempenho entre 120% e 160%, comparado ao uso de uma GPU única. Com isto, o uso de múltiplas GPUs ainda não é proveitoso para a maioria dos consumidores, mas aponta uma tendência para o futuro, à medida que a tecnologia se desenvolve e consolida.

## 5 A GeForce Series 6

Como estudo de caso, podemos analisar a GeForce Series 6 da nVidia, lançada em 2005. Analisaremos especificamente o modelo GeForce 6800 Ultra, que era a GPU mais cara da nVidia em 2005 [10].

Em termo de barramento, a GPU transfere dados com a *north bridge* a até 8 GB/s (PCI Express de 4 GB/s para cada sentido), que por sua vez se comunica com a CPU e memória RAM do sistema a 6,4 GB/s ou mais. Entre GPU e memória gráfica, a largura de banda é de até 35 GB/s (550 MHz DDR memory clock $\times$ 256 bits por clock $\times$ 2 transferências por clock).

Um diagrama de blocos básico, seguindo o modelo do *pipeline* de renderização, pode ser observado na figura 5. Cada um dos 6 processadores de vértices suporta programas *shaders* de até 512 instruções estáticas (número de instruções do programa) e 65.536 instruções dinâmicas (número de instruções executadas em um mesmo programa). Possuem também 32 registradores temporários e suporte a *branches* e *loops* (apenas recentemente adicionados aos processadores de vértices e pixels). Cada programa de vértices pode acessar até 4 diferentes texturas, apesar de isto gerar latência. Suporta também instruções exponenciais e trigonométricas, para facilitar programas que fazem transformações em polígonos.

Cada um dos 16 processadores de pixels suporta programadas *shaders* de até 65.535 instruções, estáticas e dinâmicas. Cada processador pode gerar até 4 diferentes *color buffers*. Permitem *branches* e *loops* e possuem emissão dupla de instruções.

Com isto, a GeForce 6800 Ultra tem especificações que lhe garantem desempenho muito bom. A GPU roda com um *clock* de 425 MHz, enquanto a memória gráfica roda a 550 MHz. Consegue processar 12,8 bilhões de pixels por segundo.

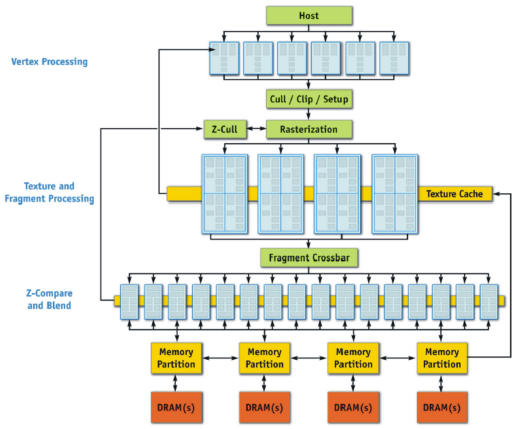


**Figura 5. Diagrama de blocos da GeForce Series 6 [10]**

## 6 Desempenho

Com tantas unidades funcionais e um *pipeline* de renderização complexo, fica difícil determinar quais aspectos impactam mais ou menos no desempenho de uma GPU. Para apresentar um estudo aprofundado sobre os aspectos de performance, usamos dos experimentos feitos em [7], que usaram uma mesma máquina alterando somente sua GPU, medindo performance através de números de quadros por segundo em jogos.

O primeiro aspecto a ser tratado é a velocidade do *clock*. A frequência em que a GPU executa tem um impacto direto (e mensurável) na performance gráfica. No exemplo em questão foram utilizadas as placas ATI Radeon modelos X1800XL e X1800XT, com arquiteturas e interfaces de memória idênticas, mas com *clocks* diferentes. Enquanto a primeira roda GPU e memória a 500 MHz, a segunda roda a GPU a 625 MHz e a memória a 750 MHz. O resultado, mostrando a importância deste parâmetro, pode ser visto na figura 6.

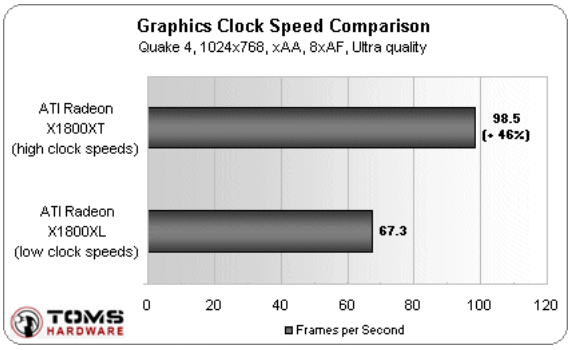


**Figura 6. Desempenho variando o *clock***

Outro aspecto considerado fundamental para desempenho é o número de unidades de *shader*. No exemplo a seguir foram examinadas a ATI Radeon X800 XL e X800 GTO. Ambas têm velocidades de *clock* e interfaces de memória idênticas. No entanto, a Radeon X800 XL tem 16 unidades de *pixel shader*, enquanto que a X800 GTO tem somente 12. A figura 7 mostra que a arquitetura do processador também tem um impacto direto na performance.

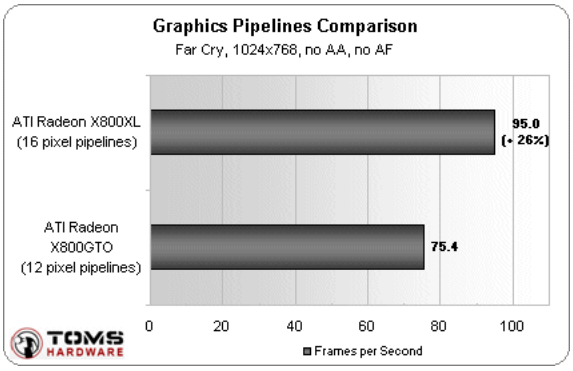


**Figura 7. Desempenho variando o número de unidades de *pixel shading***

Um aspecto muito usado por consumidores e até vendedores para comparar desempenho é o número de memória RAM da placa gráfica. No exemplo a seguir foram utilizadas duas versões da ATI Radeon X800 XL, uma com 256 MB de RAM e a outra com 512 MB. A figura 8 mostra que o tamanho da memória tem baixo impacto na performance. Isso acontece porque a maioria das aplicações utilizam somente a memória que têm disponível, sem fazer qualquer tipo de memória virtual. Dessa forma, uma quantidade baixa de memória só limita a qualidade das texturas utilizadas, mas não o desempenho.

Se a quantidade de memória não impacta tanto no desempenho, o mesmo não pode ser dito sobre a interface de memória. A figura 9 mostra comparação entre a ATI Radeon 9500 Pro e a 9700, sendo que a única diferença entre ambas é a interface com a memória, de 128 bits na 9500 Pro e de 256 bits na 9700.

Outro aspecto que possui menos impacto que o esperado é o barramento. No exemplo da figura 10 são comparadas duas versões da nVidia GeForce 6800GT, uma com barramento AGP 8X e outra com o PCI Express x16. A razão do baixo impacto do barramento é que, uma vez que ele deixa de ser um gargalo, a GPU já trabalha em sua velocidade máxima. O padrão AGP 8x ainda satisfaz requisitos de performance da maioria das placas atuais.

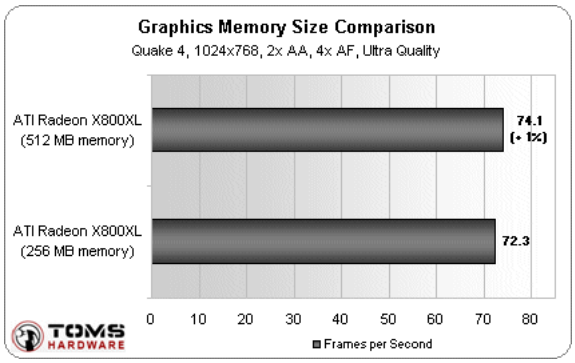


**Figura 8. Desempenho variando tamanho de memória de vídeo**

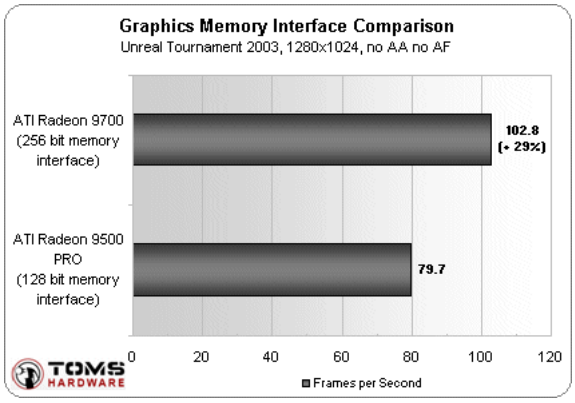


**Figura 9. Desempenho variando a interface de memória**

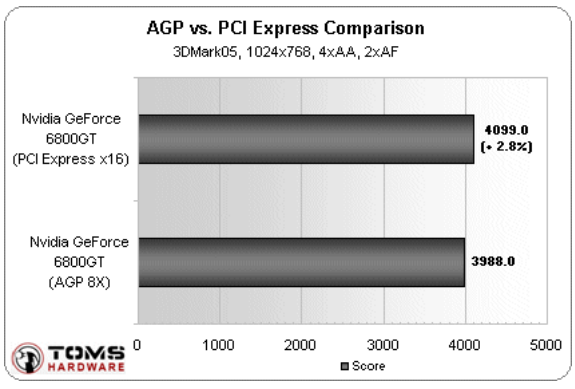


**Figura 10. Desempenho variando o barramento**

Por fim, apresentamos na figura 11 uma comparação entre o uso de uma única GPU e o uso de duas em paralelo. O modelo usado foi o nVidia GeForce 6800 Ultra. Percebe-se que, na configuração usada, o uso de duas GPUs apresentou um *speedup* de 1,46.

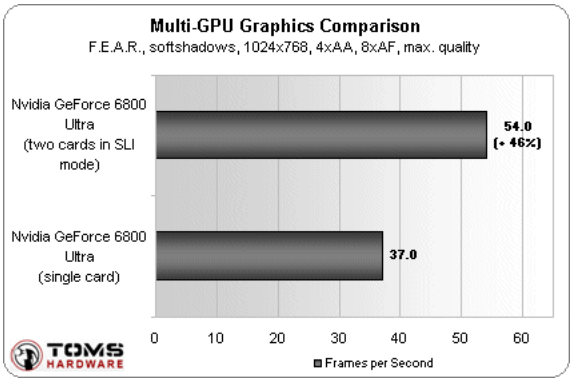


**Figura 11. Desempenho variando o número de GPUs**

## 7 Programando a GPU

O desenvolvimento de programas que utilizam a GPU não é trivial. Com o intuito de proteger seus segredos comerciais, as empresas fabricantes de GPUs disponibilizam *drivers* de placas gráficas que são como "caixas pretas", cujas implementações são desconhecidas. Toda programação passa por esses *drivers*, que são os únicos que fazem chamadas através da ISA das GPUs.

Implementar programas que acessam diretamente os *drivers* normalmente é um trabalho muito dispendioso, uma vez que a especificação dos mesmos pode mudar entre diferentes modelos de placas. Dessa forma, desenvolvedores utilizam de APIs gráficas, que provém diretivas em mais alto nível para as aplicações, e as traduzem em tempo de execução para os *drivers*. As mais famosas são o DirectX, da Microsoft, e o OpenGL, com implementação também para sistemas Linux e em código aberto.

O DirectX é atualmente a API gráfica mais usada e com isso conseguiu definir modelos de programação para GPUs. A Microsoft definiu especificações de *shader models* que fabricantes seguem em suas placas, tornando-as "*DirectX compliant*". As APIs servem também para programar os *shaders*, uma vez que as fabricantes divulgam a ISA dos processadores de vértices e pixels, mas é custoso programar diretamente em linguagem *assembly*. Com isso, o DirectX oferece uma linguagem própria para programação de *shaders*, a HLSL (*High Level Shader Language*). Outras linguagens específicas para programação de *shaders* também exsitem, como a GLSL (*OpenGL Shading Language*) e a

Cg, da nVidia. A figura 12 mostra o ISA do *pixel shader* da GeForce Series 6.

| | |
|---|---|
| ABS | absolute value |
| ADD | add |
| BRK | break out of loop instruction |
| CAL | subroutine call |
| CMP | compare |
| COS | cosine with reduction to [-PI,PI] |
| DDX | partial derivative relative to X |
| DDY | partial derivative relative to Y |
| DIV | divide vector components by scalar |
| DP2 | 2-component dot product |
| DP2A | 2-comp. dot product w/scalar add |
| DP3 | 3-component dot product |
| DP4 | 4-component dot product |
| DPH | homogeneous dot product |
| DST | distance vector |
| ELSE | start if test else block |
| ENDIF | end if test block |
| ENDLOOP | end of loop block |
| ENDREP | end of repeat block |
| EX2 | exponential base 2 |
| FLR | floor |
| FRC | fraction |
| IF | start of if test block |
| KIL | kill fragment |
| LG2 | logarithm base 2 |
| LIT | compute light coefficients |
| LOOP | start of loop block |
| LRP | linear interpolation |
| MAD | multiply and add |
| MAX | maximum |
| MIN | minimum |
| MOV | move |
| MUL | multiply |
| NRM | normalize 3-component vector |
| PK2H | pack two 16-bit floats |
| PK2US | pack two unsigned 16-bit scalars |
| PK4B | pack four signed 8-bit scalars |
| PK4UB | pack four unsigned 8-bit scalars |
| POW | exponentiate |
| RCP | reciprocal |
| REP | start of repeat block |
| RET | subroutine return |
| RFL | reflection vector |
| RSQ | reciprocal square root |
| SCS | sine/cosine without reduction |
| SEQ | set on equal |
| SFL | set on false |
| SGE | set on greater than or equal |
| SGT | set on greater than |
| SIN | sine with reduction to [-PI,PI] |
| SLE | set on less than or equal |
| SLT | set on less than |
| SNE | set on not equal |
| STR | set on true |
| SUB | subtract |
| SWZ | extended swizzle |
| TEX | texture sample |
| TXB | texture sample with bias |
| TXD | texture sample w/partials |
| TXL | texture same w/explicit LOD |
| TXP | texture sample with projection |
| UP2H | unpack two 16-bit floats |
| UP2US | unpack two unsigned 16-bit scalars |
| UP4B | unpack four signed 8-bit scalars |
| UP4UB | unpack four unsigned 8-bit scalars |
| X2D | 2D coordinate transormation |
| XPD | cross product |

**Figura 12. ISA do *pixel shader* da GeForce Series 6**

As especificações do DirectX possibilitam desenvolvedores criarem complexos efeitos visuais em linguagem de alto nível. Por exemplo, o DirectX 9 introduziu a técnica de *HDR lightning*, que possibilita uso de técnicas de iluminação com altos contrastes, aumentanto qualidade visual. Outras técnicas são também implementadas pela API, como o *anti-aliasing*, que reduz efeitos de cerrilhado em bordas de objetos, filtragem de texturas, que permitem exibições mais nítidas de texturas em ângulos oblíquos, e o uso dinâmico de conjuntos de texturas de diferentes resoluções, de forma que a aplicação se adapte a diferentes GPUs.

# 8 Tendências para o Futuro

Muito do desenvolvimento das placas gráficas está ligado ao aumento do paralelismo em *hardware*. A nova geração GeForce Series 8 da nVidia, por exemplo, está sendo lançada em novembro de 2006 e seu modelo mais caro contém 128 processadores de pixel, 32 unidades de textura e 24 unidades de operações de rasterização. Outro exemplo é a arquitetura Cell, do videogame Playstation 3 da Sony, que utiliza mútiplos *cores* de processamento tanto em sua CPU quanto na GPU, fabricada pela nVidia. Além disso, o aprimoramento das técnicas de usos de múltiplas GPUs é também uma tendência.

O desenvolvimento do DirectX também é intimamente ligado ao das GPUs, uma vez que este especifica padrões que os fabricantes de *hardware* seguem. Na sua mais nova versão, o DirectX 10, a ser lançado em janeiro de 2007, existe a especificação de *shaders* unificados, onde processadores de vértices e de pixels terão funcionalidade similar, mas com papéis distintos. Essa tecnologia já é empregada no console de videogame Xbox 360, da Microsoft e com GPU da ATI. Além disso, o DirectX 10 também especifica um terceiro tipo de *shader*, o *geometry shader*, que significa a introdução de unidades de rasterização e interpolação programáveis.

Além do desenvolvimento da arquitetura da GPU em si, a tecnologia tem gerado novos conceitos que aparecem para o mundo da computação. Os tópicos a seguir citam os principais.

## 8.1 Processadores Gráficos Embutidos

Recentemente estão sendo criadas novas GPUs com poder computacional inferior aos das melhores GPUs encontradas no mercado. Tais processadores são voltados para a utilização em dispositivos móveis inteligentes, como celulares e PDAs. As GPUs para estes dispositivos devem ser criadas embutidas no *hardware* dos mesmos e levar em conta limitações de tamanho e energia. No mercado de *laptops* já é comum o uso de GPUs próprias, como as séries nVidia GeForce Go e ATI Mobile Radeon. Agora, a tendência é o foco no mercado de portáteis como PDAs e celulares.

## 8.2 PPU - Physics Processing Unit

É sabido que os jogos são os principais "usuários" das GPUs, tendo em vista que o processamento gráfico dos mesmos é normalmente seu aspecto mais custoso computacionalmente. Nos últimos anos, porém, um novo aspecto bem complexo tem sido desenvolvido por fabricantes de jogos: a física. Com o objetivo de proporcionar experiências mais realistas, jogos têm incluído cálculos físicos, que determinam propriedades de materiais, gravidade, e outros.

Pelo fato da física envolver cálculos extremamente complexos, alguns fabricantes estão desenvolvendo a idéia da PPU, a *Physics Processing Unit*. A idéia é totalmente análoga à das GPUs, com um processador dedicado às operações de física e programado através de uma API (ou uma extensão do DirectX). Os cálculos da PPU podem incluir dinâmica de corpos rígidos e macios, detecção de colisão, dinâmica de fluidos, simulação de cabelo e tecido, análise de elementos finitos e fraturação de objetos [23].

### 8.3 GPGPU - General Purpose Graphic Processing Unit

Outro aspecto que tem despertado interesse em desenvolvedores e pesquisadores é a idéia de se usar o poder computacional de uma GPU para processamento de computação de propósito geral, ou seja, GPGPU (*General Purpose computing on Graphics Processing Units*). A técnica tem sido desenvolvida através do uso das diferentes unidades de uma GPU para efetuar outros tipos de computação. Dados tratados como texturas e uso de *shaders* para otimizar funções são só alguns exemplos de técnicas de programação GPGPU [21].

Como impulsionador dessa tendência, uma das duas maiores fabricantes de GPUs do mundo (a canadense ATI) foi recentemente comprada por uma das maiores fabricantes de CPUs no mundo (a americana AMD). Com isto, as duas recém-fundidas empresas trabalham atualmente para unir CPU e GPU em um único processador, permitindo computação de propósito geral pelas estruturas da GPU, bem como eliminando gargalos de barramento.

## 9 Conclusão

Neste trabalho foi possível estudar a importância e arquitetura das GPUs. Através de um estudo foi possível entender o conceito e aprofundar em pontos arquitetônicos dos processadores gráficos. Usando informações de desenvolvedores e fabricantes foi possível determinar aspectos fundamentais das estruturas da GPUs, ainda que estas sejam segredos comerciais.

Foi também possível fazer uma análise de desempenho, explicitando aspectos fundamentais que impactam na performance das GPUs. Analisamos ainda um exemplo específico de placa gráfica e evidenciamos aspectos de como programar esse tipo de processador. Por fim, apontamos tendências para o futuro e novas tecnologias que nasceram das GPUs.

## Referências


[1] M. Colbert. Gpu architecture & cg, 2006. http://graphics.cs.ucf.edu/gpuseminar/seminar1.ppt.

[2] S. Dietrich. Dx8 pixel shaders. In nVidia Corporation, editor, *Game Developers Conference*, 2001. http://developer.nvidia.com/attach/6514.

[3] S. Dietrich. Intro to pixel shading in dx8, 2001. http://developer.nvidia.com/attach/6677.

[4] S. Dominé and J. Spitzer. Opengl texture shaders. In nVidia Corporation, editor, *Game Developers Conference*, 2001. http://developer.nvidia.com/attach/6464.

[5] S. Green. Geforce 6 series opengl extensions, 2004. http://download.nvidia.com/developer/presentations/-2004/6800_Leagues/6800_Leagues_OpenGL_exts.pdf.

[6] T. hardware. Graphics beginners' guide, part 2: Graphics technology, 2006. http://www.tomshardware.com/2006/07/31/-graphics_beginners_2/index.html.

[7] T. hardware. Graphics beginners' guide, part 3: Graphics performance, 2006. http://www.tomshardware.com/2006/08/08/-graphics_beginners_3/index.html.

[8] R. Huddy. Introduction to vertex shaders, 2001. http://developer.nvidia.com/attach/6690.

[9] K. Huizing and H.-W. Shen. The graphics rendering pipeline, 2004. http://www.win.tue.nl/ wstahw/2IV40/pipeline2.pdf.

[10] E. Kilgariff and R. Fernando. *GPU Gems 2*. nVidia Corporation, 2005. http://download.nvidia.com/developer/-GPU_Gems_2/GPU_Gems2_ch30.pdf.

[11] D. Kirk. Geforce3 architecture overview, 2001. http://developer.nvidia.com/attach/6420.

[12] C. Maughan and M. Wloka. Vertex shader introduction, 2001. http://developer.nvidia.com/attach/6543.

[13] B. Mora. Computer graphics 2, lecture 4: Gpu programming, 2005. http://www.cs.swan.ac.uk/ csmora/Courses/CS_307/CGII-4.pdf.

[14] nVidia Corporation. Geforce256 - the world's first gpu, 1999. http://developer.nvidia.com/attach/6781.

[15] nVidia Corporation. Nvidia gpu programming guide, 2005. http://developer.download.nvidia.com/-GPU_Programming_Guide/-GPU_Programming_Guide.pdf.

[16] S. C. UK. Graphics - gpu, 2006. http://www.scan.co.uk/tekspek/view.asp?a=38&p=1.

[17] Wikipedia. 8514 (display standard), 2006. http://en.wikipedia.org/wiki/8514_(display_standard).

[18] Wikipedia. Amiga, 2006. http://en.wikipedia.org/wiki/Commodore_Amiga.

[19] Wikipedia. Atari 5200, 2006. http://en.wikipedia.org/wiki/Atari_5200.

[20] Wikipedia. Atari 8-bit family, 2006. http://en.wikipedia.org/wiki/Atari_800.

[21] Wikipedia. Gpgpu, 2006. http://en.wikipedia.org/wiki/GPGPU.

[22] Wikipedia. Graphics processing unit, 2006. http://en.wikipedia.org/wiki/Graphics_processing_unit.

[23] Wikipedia. Physics processing unit, 2006. http://en.wikipedia.org/wiki/Physics_processing_unit.