Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação

Disciplina: Computação Gráfica

AD2 - 1° semestre de 2016.

1) O são Vertex Shaders, Pixel Shaders e Geometry Shaders? Dê um exemplo de efeito que pode ser atingido por cada um. (1.0 ponto).
Vertex Shaders são programas que são programados e carregados para a GPU e interferem de alguma forma o estagio de geometria do pipeline gráfico. Efeitos possíveis com esta estratégia são texturas procedurais e diversos modelos de iluminação por vértice. Geometry shaders são programas de GPU que permitem alterar propriedades da geometria como um todo. Com eles é possível alterar a topologia ou a triangulação de uma malha, por exemplo. Pixel shaders são programas que alteram o estagio de rasterizacao. Efeitos possíveis são bump-mapping e especularidade.

2) CUDA é uma biblioteca para programar GPUs, com propósito geral. Explique conceitos básicos e exemplos do que pode ser feito.

A computação de alto desempenho ganhou uma nova ferramenta com o surgimento do CUDA (Compute Unified Device Architecture) em junho de 2007. A ferramenta CUDA permitiu a programação das placas gráficas (GPUs - Graphic Processors Unit), para computação de propósito geral, de forma mais transparente para o desenvolvedor. Contudo, estes dispositivos já estavam sendo utilizados para cálculos de proposito gerais a alguns anos. Naquela época, os dados eram ser modelados em vértices e pixels .
A utilização deste hardware se deu por ser um dispositivo dedicado, com alto poder de processamento massivamente paralelo e de baixo custo, quando comparado com as CPUs. A arquitetura das GPUs permite a construção de chips de baixo custo com e centenas de cores, atualmente há placas compostas por milhares de cores
.
No início da utilização das placas gráficas, os gráficos eram geridos através de controladores VGA, que apenas manipulavam memória de vídeo. Em 1997, alguns fabricantes começaram a incluir funcionalidades de hardware capazes de implementar parte do chamado pipeline gráfico em tempo real, tais como rasterização, mapeamento de textura e, em placas maior poder de processamento, cálculos de sombreamento. No final do ano de 2000, estes dispositivos eram capazes de processar todo o pipeline gráfico, deixando as CPUs livres da maioria das tarefas de

rasterização e passaram a equipar todos os computadores pessoais e consoles, combinando o processamento da CPU com a GPU e formando sistemas heterogêneo.
Basicamente, a arquitetura da GPU é classificada como um modelo do tipo Single Instruction, Multiple Threads (SIMT). Isto quer dizer que diversas threads executam a mesma instrução a cada ciclo de clock. Além disso, no modelo de programação do CUDA, as thread são organizadas em blocos, formando uma organização lógica das instâncias do kernel. Na parte de hardware, as GPUs são organizadas em multiprocessadores (SMPs - symmetric multiprocessors) idênticos , cada um dos SMPs é composto por centenas de cores, memórias e registradores.
A execução das threads envolve o escalonamento das mesmas em dois níveis: no primeiro nível, os blocos de threads são escalonados nos SMPs; e, no segundo nível, as threads do bloco são escalonadas nos cores. O número de SMPs pode variar de acordo com o modelo de geração e GPU. Muitas mudanças no projeto de um SMP foram feitas desde a primeira versão da GPU a fim de melhorar a performance e o consumo de energia destes processadores. Há um limite de threads e blocos que podem ser alocados em uma GPU. Atualmente o limite de threads por blocos é de 1.024 e a quantidade de blocos é de 2.147.483.647, assim é possível lançar 2.199.023.254.528 threads em uma única GPU. Por fim, conceitualmente os blocos são agrupados formando uma grid.
A GPU tem diferentes memórias, cada um com capacidade e velocidade de acesso diferente e são organizadas da seguinte forma:

- A memória global, que é a memória principal e pode ir até 12GB em arquitetura Maxwell. Todos os núcleos / threads podem acessar diretamente essa memória, mas há uma latência alta e uma baixa produção. Os dados a partir desta memória tem o tempo de vida da grelha.
- A memória compartilhada, que é uma memória de baixa latência, tem uma alta taxa de transferência e é dedicado a cada SM. Apenas segmentos pertencentes ao mesmo bloco pode aceder a esta memória e o tempo de vida é o mesmo do bloco. Quando um bloco termina, a memória compartilhada é apagada, mesmo para um bloco subsequente que irá utilizar o mesmo SM.
- A memória local, assim chamado porque o seu âmbito é local para o segmento, não por causa de sua localização física. Memória local é off-chip tornando o acesso a ele tão caro como o acesso à memória global .. Esta memória é acessível somente para um segmento específico e dados persiste apenas durante a execução de discussão.
- A textura de memória, que corresponde a uma memória só de leitura e em cache, resultando em uma leitura a partir do cache. Ele é otimizado para localidade espacial 2D, permitindo que segmentos que estão perto de usar a mesma operação de leitura para os dados correspondentes, no caso de haver coalescência. Essa memória também é capaz de interpolação de dados, operação típica na resolução de textura anti-aliasing.

- A memória constante, que também é armazenado em cache e custam uma operação de leitura a partir do cache, caso seja evitado um cache miss. É uma pequena memória, com 64 KB.

A CPU é capaz de ler e escrever na memória global da GPU, este processo ocorre através do barramento de dados que atualmente é o PCI-EXPRESS. Toda a comunicação entre o host e o device é por via deste canal.
O escalonamento dentro de cada SMP é feito em grupos de 32 threads . Assim, 32 threads consecutivas executam a mesma instrução. Ao fim da execução da última instrução, outras 32 threads consecutivas são escalonadas. Este procedimento de escalonamento é conhecido como warp. Neste sentido, recomenda-se instanciar a quantidade de threads em um bloco em múltiplo de 32. Alinhando os dados da memória com as threads de um warp, é garantido o melhor padrão de acesso à memória, ou seja, o acesso é coalescente. No procedimento de escalonamento, é criado dois índices: um índice é referente ao bloco que uma determinada thread pertence e o outro é o índice da thread dentro do bloco que ela pertence. Como 32 threads de um mesmo warp executam as mesma instruções, uma importante otimização é garantir que estas threads tenham o mesmo caminho de dados. Neste sentido, se há divergência fluxo de execução do código em uma ou mais thread de um mesmo warp , algumas threads podem ficar ociosas, serializando o execução.
Em versões mais antigas do CUDA, só era permitido criar grids com o mesmo kernel , ou seja, não existia a possibilidade de executar kernels concorrentes. Com o lançamento da arquitetura Fermi, passou a ser possível executar mais de um kernel ao mesmo tempo. Contudo, apenas a arquitetura Kepler realmente implementa a execução de kernels concorrentes. Isto porque o distribuidor de tarefas da Kepler colocada o código dentro de um pipeline único. Kernels concorrentes são escalonados internamente pela GPU e um SMP pode executar apenas o mesmo kernel . Na pratica, isto significa que cada bloco pode ter apenas threads instanciadas de um mesmo kernel e o kernel concorrente é escalonado em outro SMP.
Com o lançamento da arquitetura Kepler, uma nova tecnologia chamada Hyper-Q foi introduzida na GPU que permite diferentes programas ou threads de CPU dispararem diferentes kernels em uma mesma GPU. Até então, quando um kernel era disparado, a GPU ficava ocupada executando este kernel e não recebia outros kernels de outros programas ou threads de CPU, sendo necessário um sincronismo por barreira para que uma grid fosse finalizada antes de inicializar outra.
Um outro recurso introduzido na arquitetura Kepler é o chamado paralelismo dinâmico, que permite que sejam lançados kernels dentro de outros kernels. Até o surgimento deste recurso, apenas a CPU era capaz de lançar kernels. Este novo recurso passou a permitir que os SMPs possam chamar kernels diretamente do dispositivo. A seção de programação irá mostrar estes recursos.

3) O que são algoritmos de Frustrum Culling e Occlusion Culling (1.0 ponto).

Culling de polígonos consiste em descartar polígonos que não são visíveis para a câmera, seja por razoes de oclusao ou por razoes de estarem fora do câmera frustum. Algoritmos de Frustrum Culling eliminam polígonos fora do view frustrum e algoritmos de Occlusion Culling eliminam polígonos que estão ocultos por outros.

4) Liste o nome de 3 games engines uados na indústria e disserte brevemente sobre algum deles.
Unity - https://unity3d.com/
CryEngine - http://cryengine.com/
Unreal - https://www.unrealengine.com/what-is-unreal-engine-4

O Unity é um dos engines mais comuns da indústria. O mesmo possui uma arquitetura focada no uso de game objects, sendo necessário implementar scripts para cada game object do cenário. O Unity possui uma Asset store, onde se encontram inúmeros recursos pagos e gratuitos para serem usados nos jogos. Permite gerar jogos para todos os consoles e plataformas móveis. A programação é feita por script e pode ser em 3 linguagens diferentes: C#, JavaScript e Boo (variação do Phyton).

5) Um jogador de vídeo-game reclama que uma textura de um quadro de uma sala fica piscando quando ele se afasta demais do objeto. Explique tecnicamente para ele o que é que está acontecendo. (1.0 ponto).

O problema que acontece é conhecido como alias. Acorre porque a textura é armazenada contém um número finito de valores discretos de alguns imagem original, ou seja, a uma perda da informação. Ao ser aplicada sobre uma primitiva do OpenGL, por exemplo. Ao realizar a varredura de conversão da textura para o polígono, as suas coordenadas de textura são mapeados em coordenadas (u, v) e a textura é re-amostrados usando essas coordenadas. Assim, no caso em que o polígono é encolhido devido à perspectiva, que irá abranger apenas alguns pixels. Isso resultará em apenas um dos pontos de amostragem poucos espalhados por toda a imagem de textura e, portanto, causando este efeito indesejado.

6) Como se otimizam os cálculos de colisão em jogos?

Cálculos de colisão são otimizados usando-se volumes envolventes: ao invés de fazer os cálculos malha-malha, realizam-se cálculos apenas com os volumes envolventes (caixas, esferas, cápsulas). Estruturas espaciais, como Octrees e Quad-Trees também podem ser usadas para melhorar a busca de objetos próximos e colisiveis.

7) Diferencie o conceito de Iluminação global e iluminação local. (1 ponto)

Iluminação global é aquela que leva em consideração todos os raios de luz do cenário, incluindo aqueles derivados de múltiplas reflexões e difrações. Iluminação local é aquela que apenas considera os raios de luz derivados diretamente da fonte de luz.

8) O que é o algoritmo de radiosidade? Explique sucintamente seu funcionamento (1.0 ponto)

É um modelo de iluminação que leva em conta iluminação indireta, além da direta. Seu funcionamento se baseia em dividir a malha em patches, de onde se estimará quanto de energia luminosa é emitida por cada um, levando esta energia em conta no cálculo da iluminação e cada polígono ou ponto.

9) Porque as texturas procedurais podem ser geradas nas placas gráficas programáveis? (1.0 ponto)

As texturas procedurais precisam gerar dados para cada pixel, que deve receber uma cor de uma textura. Na razterizacao convencional, a simples interpolação não e' capaz de gerar dados individualizados para cada texel. As placas programáveis permitem gerar dados individuais para cada pixel, interferindo no processo de rasterizacao. Pelo fato de poder executar uma função por pixel, estas placas permitem criar dados de textura pixel a pixel.

10) O processo de rasterizar polígonos pode ser trivialmente paralelizável, uma vez que cada triângulo pode ser tratado individualmente e separadamente. Entretanto, no final do pipeline, pode haver problemas de sobreposição dos mesmos. Explique como isto é tratado e porque isto não interfere no paralelismo (1.0 ponto)

Utiliza-se o algoritmo de Z-Buffer. Este algoritmo consiste em criar uma memória de tamanho equivalente ao frame-buffer. Sempre que um pixel referente a um triangulo for pintado no frame-buffer, será escrito no frame-buffer a profundidade do mesmo. Caso já haja uma profundidade escrita anteriormente no Z-Buffer, antes de pintar o pixel será feita uma consulta se este novo pixel possui profundidade maior ou menor. Caso seja menor, permite-se a sua escrita, em cima

do anterior. Neste caso atualiza-se o valor do Z-Buffer com o valor deste novo pixel. Caso o valor de profundidade seja menor, impede-se a escrita deste pixel no frame-buffer, pois já há um pixel mais próximo pintado previamente. Como esta memória é acessada sem ordem, não há problemas de que vários triângulos sejam desenhados ao mesmo tempo. É importante apenas não permitir que dois processos queiram escrever dados no mesmo pixel ao mesmo tempo.