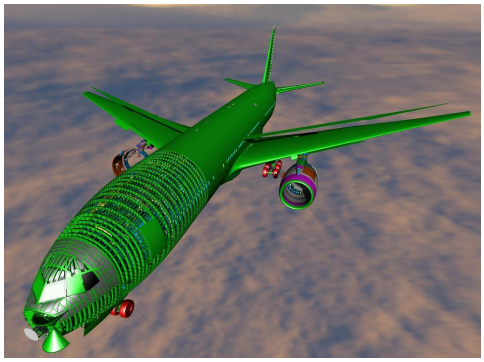
1
Introdução

1.1
Motivação

As constantes buscas pela visualização de modelos maiores e com maior nível de detalhes têm motivado as indústrias de *software* e *hardware* a desenvolver novas soluções que os viabilizem. Mesmo não tendo um valor preciso, a literatura descreve que a interação em um sistema de visualização deve prover no mínimo 30 (*fps*)[1]. Quando estamos tratando de modelos massivos como o Boeing 777 1.1(a) (extraída de [20]) e Sunflowers 1.1(b) (extraída de [73]), que possuem uma enorme quantidade de polígonos e texturas, manter esta taxa de renderização se torna um desafio.



1.1(a): 350 milhões de triângulos.

1.1(b): 1 bilhão de triângulos.

Figura 1.1: Modelos Massivos.

Um dos grandes problemas é que a complexidade dos modelos cresce na mesma velocidade ou acima da *Lei de Moore*[2]. Outro fator que deve ser levado em conta é que mesmo que o poder de processamento computacional tente acompanhar a Lei de Moore, outros fatores influenciam no desempenho do

[1] fps - *frames per second* ou quadros por segundo mede a quantidade de imagens que são atualizadas no *display* no intervalo de 1 segundo.

[2] A Lei de Moore surgiu a partir de uma observação feita por Gordon Moore, co-fundador da Intel, em 1965, que o número de transistores por polegada quadrada estava dobrando a cada 18 meses desde a introdução dos circuitos integrados. Ele previu que esta tendência iria continuar nas próximas décadas, o que acabou tornando-se verdade.[64]

sistema, como tempo de acesso ao disco rígido e à memória, que não seguem a mesma evolução que os microprocessadores.

Diferentemente das CPUs (*Central Processing Units*), as GPUs (*Graphics Processing Units*) são processadores dedicados para a parte gráfica e são muito paralelizados. Nos últimos anos o poder de processamento das placas gráficas sofreu grande aumento, superando a lei de Moore. A cada nova geração de placas, novos recursos e algoritmos são incorporados aumentando o seu desempenho e tornando viável a renderização de cenas cada vez maiores.

Enquanto as CPUs atualmente podem trabalhar com oito núcleos simultaneamente, as GPUs mais modernas chegam a ter 320 *shaders processors* em paralelo. Uma rápida comparação no número de *flops* [3] das CPUs e GPU já dá uma boa ideia do poder de processamento das GPUS. Enquanto uma CPU Core 2 Extreme QX9650 chega a 48 *gigaflops* [4], uma GPU GeForce GTX 280 consegue 933 *gigaflops*.

A partir de 2001 as GPUs sofreram uma grande mudança, tornando alguns de seus estágios programáveis (antes tais estágios eram implementados diretamente em *hardware* sem nenhum tipo de acesso externo). Com isso, vários algoritmos novos foram desenvolvidos em diversas áreas. Uma área que surgiu com o acesso aos estágios do *pipeline* da placa gráfica aliada ao seu poder de processamento foi o *GPGPU* (*General-purpose computing on graphics processing units*).

Mesmo com a constante evolução das placas gráficas, a demanda por modelos com geometrias mais complexas, iluminação global e resolução de *display* cada vez mais alta ainda superam essa evolução. Por essas razões, muitos algoritmos têm sido desenvolvidos juntamente com a evolução do *hardware*. Alguns algoritmos tentam descartar objetos que estejam muito longe do observador utilizando técnicas como *fog* ou até mesmo eliminando-os, outros aumentam o nível de detalhes dos objetos gradativamente à medida que os objetos ficam mais próximos do observador (LOD-*level of detail*). Apesar de válidos em muitas aplicações, estas técnicas influenciam, mesmo que de forma mínima, na imagem final, o que não é desejado em várias situações.

Este trabalho vai focar em um dos algoritmos de visualização conhecido como *frustum culling*, tentando aplicar as técnicas de *GPGPU* para acelerar o descarte de objetos que não estejam visíveis.

[3] flops - *Floating point Operations Per Second* é uma medida de performance na computação, especialmente na área de cálculos científicos que utiliza cálculos em ponto flutuante.

[4] gigaflops- $10^9 flops$.

1.2 Objetivo da dissertação

Um dos primeiros objetivos deste trabalho é fazer uma investigação do estado da arte do algoritmo de *frustum culling*. A partir desta investigação, pretende-se acoplar a ele técnicas de aceleração a fim de obter o melhor algoritmo possível em CPU.

Como atualmente só se conhece implementação deste tipo de algoritmo em CPU e as GPUs estão se tornando cada vez mais poderosas e acessíveis para programação, um segundo objetivo deste trabalho é fazer uma implementação desse algoritmo em GPU e compará-la com o que foi obtido em CPU, analisando os pontos positivos e negativos.

Outro objetivo é implementar o descarte de objetos paramétricos diretamente em GPU. Tais objetos são conhecidos como *gpu primitives*. Diferentemente dos modelos naturais onde um objeto é representado por um conjunto de vértices, as *gpu primitives* são representadas por informações paramétricas que permitem sua renderização diretamente na placa gráfica. Aliado ao fato de que as *gpu primitives* são renderizadas diretamente na GPU, o cálculo de descarte será inserido neste *pipeline* de três maneiras diferentes e comparado com a implementação da CPU.

O último objetivo deste trabalho é implementar um *frustum culling* híbrido utilizando a GPU em momentos oportunos para determinar quais os objetos estão visíveis pela câmera. Tais gargalos podem diminuir consideravelmente o desempenho da aplicação dependendo da cena quando tratados apenas em CPU. Esta parte do trabalho tentará identificar os momentos oportunos para uso da GPU e melhorar o desempenho para que a interação não seja afetada.

1.3 Trabalhos relacionados

Os principais trabalhos relacionados a esta dissertação podem ser separados em dois grandes grupos. O primeiro engloba os estudos feitos em cima do algoritmo de *frustum culling* e técnicas de aceleração a ele aplicadas. Dentre eles, pode-se destacar [3], que desenvolveu algoritmos que aceleram os cálculos de descarte. Outro trabalho desenvolvido também por Assarsson *et al.* [4] tenta aplicar as técnicas de *frustum culling* em máquinas multiprocessadas. [48] removeu a necessidade da utilização de uma pilha para realizar o percurso da hierarquia. Mais recentemente [53] implementou uma nova forma de fazer descarte de objetos sem a necessidade de fazer extração de matrizes e cálculo de planos a cada *frame*. Além dos trabalhos citados, existem outros

trabalhos relacionados a esta dissertação que serão citados e discutidos com mais detalhes no capítulo 3.

O segundo grupo de trabalhos está relacionado aos algoritmos desenvolvidos para GPU. Como não é conhecida nenhuma implementação do algoritmo de *frustum culling* em GPU, os trabalhos relacionados deste grupo foram implementados para outros fins, porém a ideia básica foi aproveitada. Um exemplo é o trabalho de Thrane e Simonsen *et al.* [48], já citado anteriormente, que desenvolveu um percurso de hierarquia em GPU a priori para acelerar algoritmos de *ray tracing*. Um dos trabalhos que serviram de base para uma parte deste trabalho foi a tese de doutorado de Toledo [70], onde são apresentadas as *gpu primitives*. O Capítulo 5 explicará com mais detalhes esses trabalhos.

1.4 Organização da dissertação

Esta dissertação está dividida em sete capítulos. O próximo capítulo faz uma breve revisão sobre os algoritmos de determinação de visibilidade, dando destaque para os algoritmos de *culling*. Além desta breve revisão sobre os principais algoritmos, eles serão classificados segundo suas características.

O Capítulo 3 inicia apresentando o algoritmo de *frustum culling* básico. Este capítulo analisa algumas técnicas de aceleração utilizadas para melhorar o desempenho do algoritmo.

O Capítulo 4 tem por finalidade avaliar a implementação dos algoritmos e técnicas de aceleração levantadas no Capítulo 3 e ao final utilizar as que obtiveram melhor desempenho para servir de base de comparação com os algoritmos desenvolvidos em GPU.

O Capítulo 5 analisa a evolução do poder de processamento das placas gráficas ao longo dos tempos. Depois são propostos e implementados três algoritmos de *frustum culling* em GPU. São feitas análises e comparações em cima do algoritmo conseguido em CPU no capítulo anterior.

O Capítulo 6 explora a combinação entre o melhor dos mundos conseguido em CPU e em GPU, utilizando cada qual nos momentos onde conseguem melhor desempenho. Os resultados desta abordagem são comparados e analisados com as outras abordagens implementadas.

O Capítulo 7 analisa os resultados obtidos em todas as abordagens e comenta se os objetivos do trabalho foram alcançados. Também são sugeridos trabalhos futuros na área e as contribuições deixadas com este.