7
Conclusão e Trabalhos futuros

7.1
Conclusão

Este trabalho fez um levantamento das principais formas do algoritmo de *frustum culling*, indicando o seu estado da arte juntamente com técnicas de aceleração visando melhoria de performance. Como resultado, o algoritmo de extração dos planos do *frustum*, aliado à técnica de aceleração obteve melhor resultado. Na maioria dos testes as técnicas foram: utilização de hierarquia, percurso sem a utilização de pilha, *plane-coherency*, *octant-test* e teste com apenas dois vértices da AABB. Em apenas dois modelos (P-38 e P-50) o teste entre a AABB da câmera e o volume envolvente trouxe ganho de performance.

Com a inserção de hierarquia, as técnicas de testes entre a AABB da câmera e os volumes, *plane-coherency*, *octant test*, hierarquia e percurso sem a utilização de pilha compõem o estado da arte do algoritmo de *frustum culling* em CPU. A partir deste algoritmo foram exploradas novas formas de descarte implementadas em GPU. A motivação para tal foi o grande poder de processamento das placas de vídeo e os seus estágios cada vez mais configuráveis.

O *frustum culling* em GPU foi explorado em cima de dois tipos de modelos: as *GPU primitives* e os modelos com malha triangular. Sem a utilização de hierarquia, os desempenhos dos algoritmos em GPU, para os dois tipos de modelos, sem nenhuma técnica de aceleração obtiveram performance muito superior ao implementado apenas em CPU sem hierarquia. O melhor algoritmo implementado para as *GPU primitives* teve ganho de três vezes em média, enquanto para os modelos genéricos o ganho chegou a dez vezes. Porém, quando a hierarquia é adicionada ao algoritmo em CPU, esse algoritmo supera o da GPU.

A fim de unir os dois, CPU e GPU para executar o algoritmo de *frustum culling*, surgiu o *frustum culling* híbrido. Basicamente a ideia foi utilizar cada um deles onde apresenta melhor desempenho, aumentando assim a performance do algoritmo em geral. A melhor implementação conseguida

envolveu a chamada da GPU quando o tempo de processamento da CPU excede o tempo de *download* de informações (gargalo do algoritmo em GPU). O *frustum culling* híbrido contribuiu para um ganho de performance em todos os testes, sendo mais evidente no modelo do Boeing onde a melhoria foi de 76.75% na performance quando comparado com o melhor algoritmo implementado somente em CPU.

Com o ganho de performance alcançado na maioria dos casos de testes apresentados, os objetivos desta dissertação foram alcançados, deixando alguns assuntos para serem trabalhados no futuro, conforme visto na próxima seção.

7.2 Trabalhos futuros

O algoritmo de radar pode gerar falsos positivos, ou seja, volumes envolventes que estão fora do *frustum* são tratados como visíveis. Uma análise mais detalhada desse caso pode ser feita comparando com os outros algoritmos.

Os algoritmos multiprocessados implementados não corresponderam ao esperado na teoria, principalmente o que utiliza *tasks*. Um estudo melhor sobre essa arquitetura pode melhorar o algoritmo em CPU. Outra possibilidade de analise seria a utilização de POSIX Threads [55].

Um dos grandes problemas dos algoritmos de *frustum culling* implementados em GPU é a necessidade de guardar muitos volumes envolventes na memória da placa gráfica, o que pode se tornar um problema. Para as *GPU primitives*, os volumes envolventes podem ser determinados *on-the-fly*, minimizando assim o uso da memória da GPU. Quando há hierarquia, esse problema é agravado uma vez que o número de volumes envolventes aumenta. As AABBs necessitam de 32 *bytes*, para guardar estas informações além do seu id e o *escape index* do nó, onde são necessários dois *texels*. Uma possível solução para diminuir a memória utilizada é a codificação dos nós da hierarquia utilizando quatro ou oito *bytes*, um *texel*, desenvolvido por Mahovsky *et al.* [41].

Outra linha que pode ser seguida é a melhoria do algoritmo de descarte proposto por Reshetov [58], para dar suporte a seis planos e fazer a determinação de colisão entre o *frustum* e o volume envolvente, testando o algoritmo em CPU e GPU.

Testes podem ser feitos para acelerar o cálculo de descarte na GPU utilizando as técnicas de aceleração propostas e implementadas na CPU.

Uma questão ainda em aberto é determinar uma heurística ideal de entrada, permanência e saída da GPU para modelos diversos.

Com a evolução de múltiplas placas gráficas, seria interessante destinar uma placa gráfica para a renderização e outra para os algoritmos de *frustum*

culling, tentando evitar que o algoritmo de *frustum culling* em GPU dispute recursos com a renderização.

Por fim, a estrutura híbrida implementada nesta dissertação pode ser testada em outros algoritmos como, por exemplo, na detecção de colisão, visando ganho de performance.