6
Frustum culling híbrido

Com a evolução rápida do poder de processamento das placas gráficas, muitos algoritmos têm migrado suas partes ou todas suas rotinas para GPU. Porém, vetorizar alguns problemas nem sempre é uma tarefa fácil e em alguns casos não vale a pena. Este trabalho visou até agora mostrar o estado da arte do *frustum culling* em CPU e maneiras de torná-lo portável para a GPU. Este capítulo propõe algumas heurísticas para determinar o momento ideal, onde a CPU possui pior performance, para utilização da GPU a fim de acelerar o algoritmo de determinação dos objetos visíveis.

6.1
Heurísticas

O estado da arte do algoritmo de *frustum culling* em CPU obtém boa performance em cenas pequenas e médias. O grande impacto no seu desempenho ocorre em cenas com grande número de objetos, comum em modelos CAD. Isso ocorre devido à grande quantidade de nós internos da árvore e consequentemente o excessivo número de testes contra o *frustum*. Os resultados conseguidos em GPU confirmaram o grande poder de processamento da placa gráfica quando os problemas são tratados em paralelo. A utilização de hierarquia comprometeu a execução dos algoritmos desenvolvidos em GPU, uma vez que o percurso deve respeitar uma ordem fixa e a existência de limitações no estágio de *geometry shader*. Mesmo tendo seus pontos fracos, tanto a implementação em CPU quanto em GPU apresentam boa performance em situações diferentes.

A fim de minimizar o impacto da inserção do *frustum culling* no *pipeline* das aplicações que visam a manipulação de modelos massivos, foi implementado o *frustum culling* híbrido. A ideia básica é inicialmente fazer descarte de primitivas utilizando o melhor algoritmo conseguido em CPU e quando for identificado que a quantidade de cálculos está elevada ao nível de influenciar a performance da aplicação, a GPU assume o controle do algoritmo de *frustum culling* de forma que seu tempo de processamento seja menor que o da CPU. As grandes questões dessa abordagem são decidir quando é o momento certo

de tratar os dados na GPU, como fazer com que todo o poder da GPU seja utilizado e quando retornar os cálculos para a CPU. Esses assuntos serão abordados nas Seções 6.1.1 e 6.1.2.

6.1.1 Identificação do momento ideal

A identificação do momento para a utilização da GPU é crucial para melhorar a performance da aplicação, pois se não for bem escolhido pode diminuir o desempenho. Idealmente a CPU deve controlar o processamento do algoritmo de *frustum culling* na maior parte do tempo possível, isso porque possui hierarquia e técnicas de otimização que diminuem o número de cálculos a serem feitos. A medida que os cálculos vão crescendo, a única saída da CPU para acelerar a performance do algoritmo é adotar alguma técnica mais conservativa que acabaria enviando dados não visíveis para serem renderizados. A transição de *hardware* para execução do algoritmo servirá de alternativa para o baixo desempenho da CPU em momentos de muitos cálculos e ao mesmo tempo evitar que objetos não visíveis sejam processados desnecessariamente. As Figuras 6.1 e 6.2 fazem uma comparação das performances dos melhores algoritmos conseguidos em CPU e em GPU.

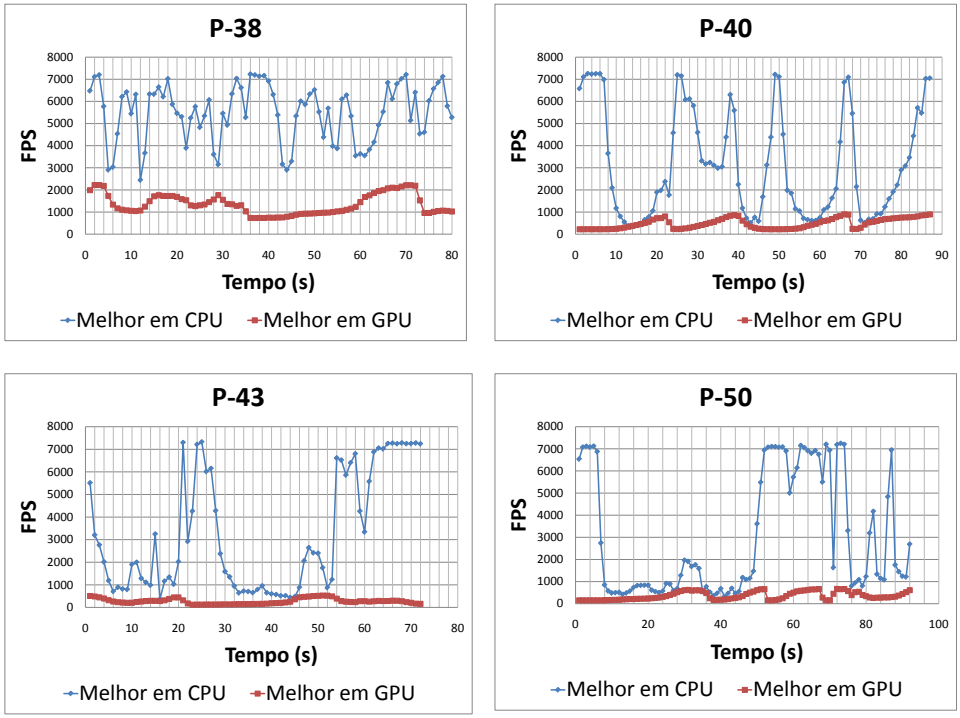

Figura 6.1: Melhores caminhos de câmera em CPU e GPU nas plataformas.

Os melhores resultados com cada um dos *hardwares* executando o algoritmo de *frustum culling* separadamente, mostra que a CPU sempre obtém

melhor performance que a GPU. Apenas em alguns momentos dos modelos maiores a performance da GPU alcança o desempenho obtido pela CPU. Vale lembrar que o algoritmo em CPU possui hierarquia e várias técnicas de aceleração aclopadas ao algoritmo de *frustum culling*, enquanto a GPU só tem a técnica de aceleração onde dois vértices são testados contra o *frustum*, processando todos os volumes envolventes sem hierarquia.

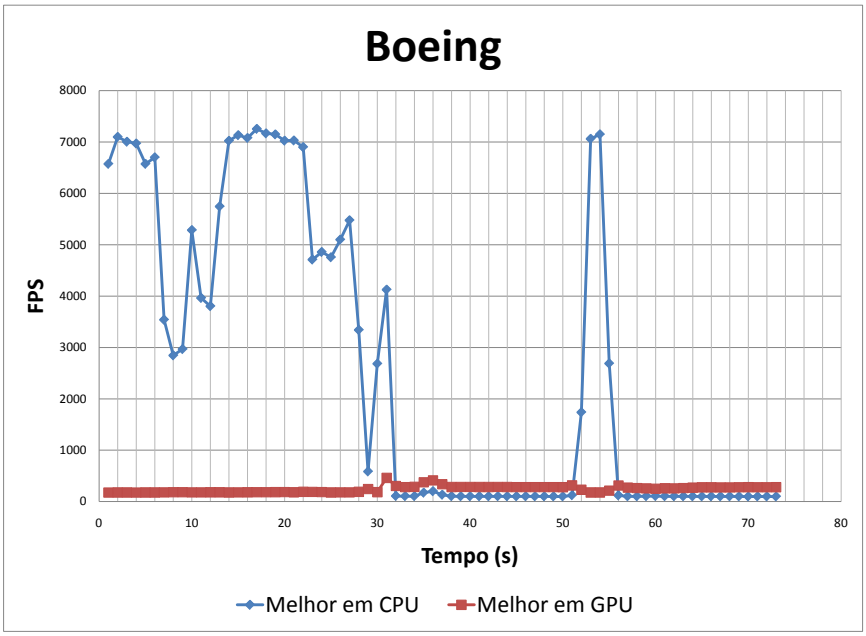

Figura 6.2: Melhores caminhos de câmera em CPU e GPU no Boeing.

No caso do Boeing, a GPU, mesmo processando todos os volumes envolventes, supera o desempenho da CPU em vários *frames*. Tanto no caso das plataformas quanto no caso do Boeing, existem várias quedas de desempenho no algoritmo puramente em CPU. Esses momentos de queda de desempenho da CPU, seriam os momentos mais adequados para a migração do *frustum culling* para a GPU.

Foram experimentadas quatro formas para identificação do melhor momento para utilização da GPU, descritas com mais detalhes abaixo.

1. Altura da hierarquia - identifica gargalos a partir do momento em que o percurso da hierarquia ultrapassa uma determinada altura.

2. Número de interseções - como uma das causas do gargalo do algoritmo ocorre quando há excessivo número de interseção entre os volumes envolventes e o *frustum*, este identifica os gargalos a partir de um certo número de interseções.

3. Porcentagem de nós processados - neste caso, a identificação é feita a partir do momento que o número de nós processados ultrapassa um determinado valor.

4. Tempo de processamento - a identificação neste caso é feita através do tempo de processamento gasto para realizar o percurso da hierarquia.

A Seção 6.2 discute o desempenho de cada uma destas heurísticas e identifica qual delas foi escolhida para ativar o uso da GPU. Tendo os momentos de transição dos cálculos da CPU para a GPU identificados, falta tornar o algoritmo vetorizável a partir dos resultados obtidos na CPU e tirar proveito do poder de processamento da GPU, o que será discutido na próxima seção.

6.1.2 Paralelização do algoritmo

Os grandes problemas de realizar o percurso da hierarquia totalmente em GPU são:

1. O percurso tem que seguir uma ordem, o que dificulta a paralelização do algoritmo.

2. Envolve muitos acessos a textura, que em modelos massivos diminuem a performance do algoritmo.

3. Limitação no *output* do estágio de *geometry shader*, que não permite escrita de todos os resultados em memória quando são utilizados modelos massivos.

Com isso, a ideia é processar apenas uma parte dos nós em paralelo, uma vez que se todos os nós forem processados, a peformance global vai diminuir, como foi visto nos gráficos da seção anterior. Para tal é necessário que toda a hierarquia esteja disponível na GPU para eventuais consultas. Além de fazer o processamento em paralelo, é importante que o resultado fornecido pela GPU contenha apenas as geometrias visíveis, o que envolve o percurso da hierarquia em GPU. Como a ideia é processar apenas uma parte da hierarquia e em paralelo, os acessos a textura são diluídos entre vários nós, o que não acontecia no percurso completo da hierarquia em GPU. Além disso, o processamento dos nós em paralelo contorna a limitação do *geometry shader* de escrever em memória, pois cada nó processado poderá escrever até 1024 resultados na placa utilizada nos testes.

Para viabilizar a utilização da GPU em alguns momentos, juntamente com a CPU, foi explorada a ideia da coerência temporal, que funciona da seguinte forma: se em um determinado *frame* x de processamento em CPU é identificada a necessidade (gargalo em CPU) de uso da GPU, os índices dos últimos nós processados em CPU são enviados para a GPU para que no *frame* $x+1$ eles sejam processados em paralelo na GPU. Assim, pela coerência temporal, é bem possível que os nós processados em GPU não necessitem fazer muitas consultas à textura na operação de percurso pela hierarquia. A Figura 6.3 mostra o esquema do algoritmo.

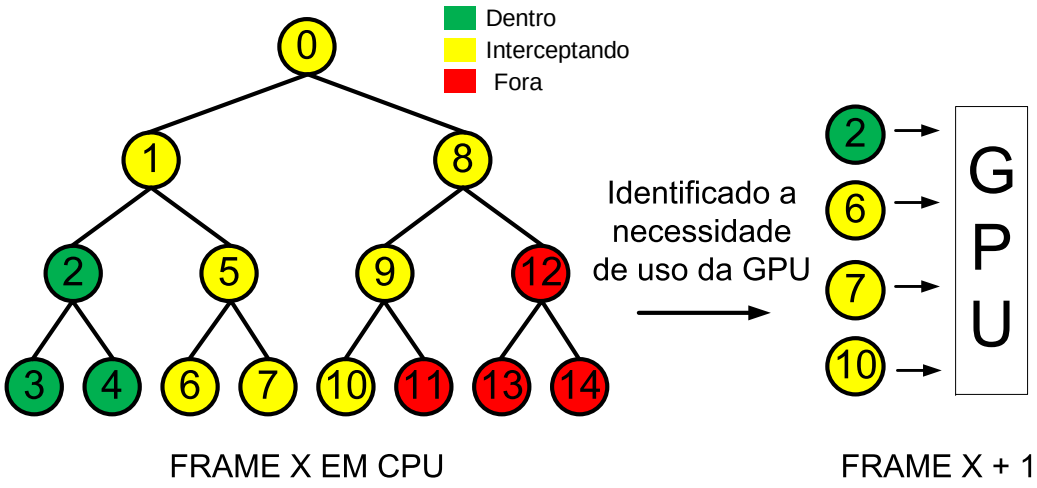

Figura 6.3: Esquema do algoritmo de frustum culling híbrido.

Na Figura 6.3 os nós 2, 6, 7, 10 foram os últimos nós classificados como visíveis, ou seja, suas geometrias participam da imagem final. Uma vez que os nós estejam sendo processados em GPU, os resultados emitidos se tornarão a entrada do *frame* seguinte. Desta forma é mantida a coerência temporal entre *frames*, diminuindo assim o percurso na GPU, o acesso elevado a textura e possíveis erros gerados pela limitação do *geometry shader*. Caso o resultado de um dos nós for negativa, ou seja, fora do *frustum* de visão, o nó pai correspondente deve ser adicionado como *input* de processamento no próximo *frame*. Por exemplo, se no *frame* x o nó 10 for processado em GPU e seu resultado der fora do *frustum*, no *frame* $x+1$, o nó 9 deve ser processado.

Resolvido o problema de entrada e manutenção do algoritmo em GPU, é necessário também monitorar o momento para deixar de usar a GPU. Isso porque se o algoritmo rodar apenas na GPU, os momentos em que a CPU processa a hierarquia mais rápida serão desperdiçados. Esses momentos ocorrem porque o retorno dos resultados da GPU para a CPU demora mais tempo que a própria execução do algoritmo em GPU tornando-se assim o gargalo quando a GPU é utilizada. Um exemplo dessa situação pode acontecer

quando a câmera está interceptando muitos nós em um determinado *frame* e nos *frames* seguintes nada é observado. Para que a inserção da GPU no *pipeline* não diminua a performance do algoritmo, foram desenvolvidos estágios de transição entre CPU e GPU, como pode ser observado na Figura 6.4.

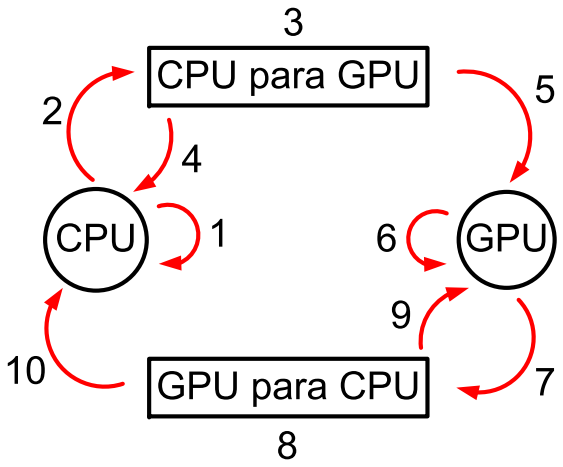

Figura 6.4: Possíveis estados do frustum culling híbrido.

1. Inicialmente o descarte de volumes envolventes é feito no melhor algoritmo conseguido em CPU, sendo verificado a cada iteração se ocorreu algum gargalo no algoritmo. Caso não ocorra, o algoritmo roda apenas em CPU.

2. Caso ocorra gargalo, o estado corrente muda para o número 3 (CPU para GPU).

3. Neste estágio é verificado se nos n *frames* consecutivos o gargalo ainda permanece na CPU.

4. Caso o gargalo na CPU não permaneça, o valor n é zerado e o estado corrente volta para a CPU.

5. Caso o gargalo na CPU permaneça, o contador deste estágio é zerado e a GPU assume o controle do algoritmo.

6. A GPU permanecerá com o controle do algoritmo por m *frames*.

7. Depois de m *frames* processados em GPU, ocorre uma transição para o estágio "GPU para CPU", onde a CPU retoma o controle do algoritmo.

8. Neste estágio é verificado se no *frame* atual ainda ocorre gargalo na CPU.

9. Caso ainda haja gargalo na CPU o valor m é incrementado e a GPU retoma o controle do algoritmo.

10. Caso não haja mais gargalo a CPU retoma o controle do algoritmo.

O que estamos chamando aqui de gargalo, e os valores de n e m *frames* utilizados são discutidos na seção seguinte.

6.2 Implementação

Para que o esquema do *frustum culling* híbrido funcione bem, os gargalos em CPU e o momento de saída da GPU devem ser bem estimados. Várias abordagens foram tentadas para determinar o momento ideal de entrada na GPU em todos os modelos.

1. A identificação por altura em todos os modelos gerou muitos falsos positivos. Isso se deve ao fato de que nem sempre que o percurso em CPU atingia uma certa altura, um gargalo é identificado.

2. Utilizando o número de interseções, a determinação se mostrou eficiente em alguns modelos, porém a determinação de um valor que se encaixe bem para as diferentes hierarquias dificultou a utilização deste.

3. A identificação por porcentagem dos nós processados não obteve bom desempenho pela dificuldade de determinação do valor ideal para as hierarquias.

4. A identificação por tempo de processamento foi a heurística que melhor se enquadrou em todos os modelos. A ideia é que seja identificado um gargalo em CPU quando o seu tempo de processamento ultrapassar o de *download* dos resultados da GPU para CPU no pior caso de testes. O pior caso dos testes foi realizar *download* de todos os *ids* dos volumes envolventes do Boeing com a duração de $2x10^{-3}$ segundos. Este valor foi reduzido para $3x10^{-4}$ segundos para aumentar as identificações dos momentos de transição na maioria dos modelos.

Os valores definidos para entrada e saída da GPU foram cinco e dois respectivamente, ou seja, se o tempo de processamento do algoritmo em CPU ultrapassar o valor máximo ($3x10^{-4}$ segundos) por cinco *frames* consecutivos, a GPU é ativada para realização dos cálculos. Inicialmente a GPU processa 500 *frames*, depois desse tempo a CPU retoma o algoritmo e se em dois *frames*, o tempo de processamento ainda estiver acima do valor máximo o tempo de permanência em GPU é dobrado e a GPU volta a processar o algoritmo.

As Figuras 6.5 e 6.6 ilustram a performance do algoritmo de *frustum culling* híbrido, com as configurações levantadas anteriormente, comparado com o melhor algoritmo conseguido utilizando apenas a CPU.

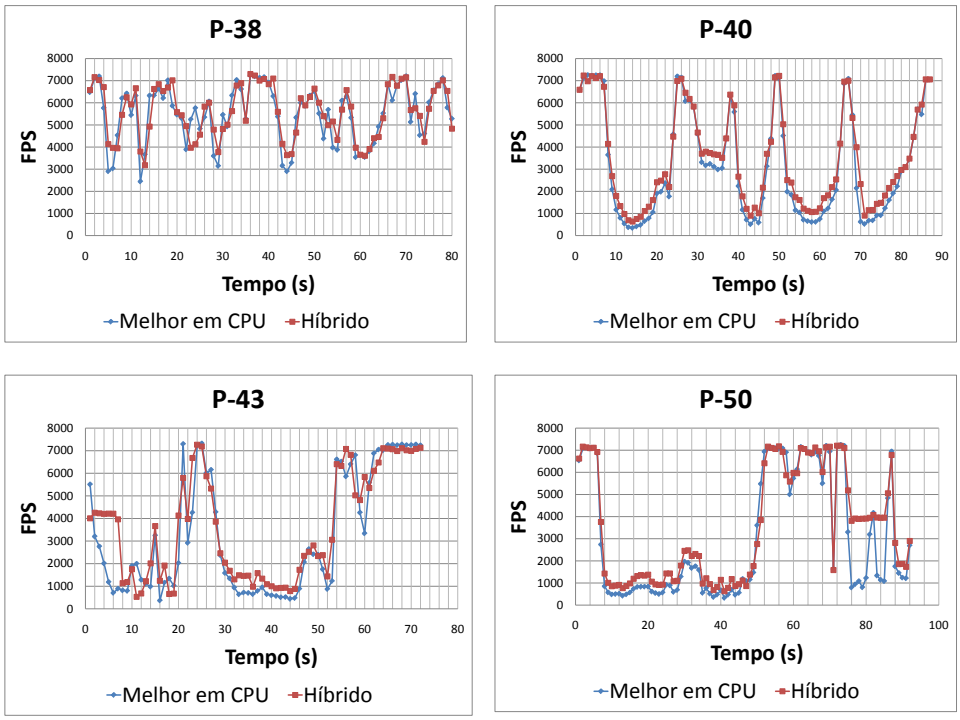

Figura 6.5: Resultados do frustum culling híbrido nas plataformas.

Em todos os testes a presença do *frustum culling* híbrido com os parâmetros especificados sempre melhorou a performance do algoritmo. Em alguns casos como na P-38 a identificação de gargalos não foi muito boa, porém no caso do Boeing, as melhorias foram notórias. Em alguns *frames* é possível notar que a performance fica abaixo do melhor algoritmo em CPU. Isto talvez se deva à perda da coerência temporal utilizada na otimização de *plane-coherency*, ou simplesmente erros de medição.

A Tabela 6.1 mostra com mais detalhes o ganho de performance da abordagem híbrida.

6.2.1 Frustum culling híbrido com a renderização habilitada

Como foi visto anteriormente, em todos os modelos de testes, a abordagem híbrida obteve melhor resultado que as outras estudadas. Porém estes resultados foram obtidos com a renderização desabilitada, ou seja, sem produzir imagem. Quando habilitamos a renderização, o desempenho da aplicação como um todo diminui, pois a placa gráfica passa a ter duas tarefas que em

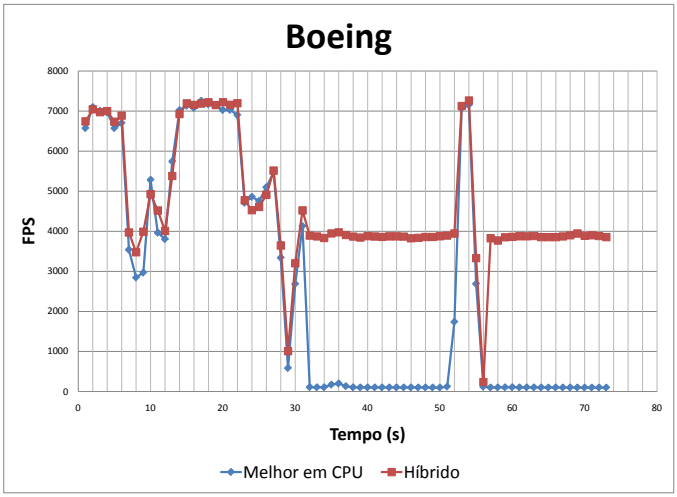

Figura 6.6: Resultados do frustum culling híbrido no Boeing.

Método	Frames Processados	FPS Mínimo	FPS Máximo	Média do Caminho
Melhor CPU na P-38	440594	2368	7241	5494.581
Híbrido na P-38	**451997**	**2756**	**7301**	**5615.358**
Melhor CPU na P-40	274740	320	7263	3143.083
Híbrido na P-40	**305221**	**596**	**7266**	**3490.873**
Melhor CPU na P-43	237059	358	**7334**	3265.365
Híbrido na P-43	**261935**	**527**	7270	**3627.657**
Melhor CPU na P-50	278401	311	**7256**	3022.32
Híbrido na P-50	**323132**	**578**	7241	**3505.028**
Melhor CPU no Boeing	191234	96	7259	2605.687
Híbrido no Boeing	**275646**	**280**	**7300**	**4629.561**

Tabela 6.1: Comparação dos resultados entre melhor algoritmo em CPU e Híbrido.

conjunto acabam tornando-se o gargalo da aplicação. Quando a renderização é chamada, a placa gráfica está ocupada com os cálculos do *frustum culling*, tendo que esperar até que eles terminem. A Figura 6.7 mostra a comparação entre os algoritmos de *frustum culling* com a renderização ligada.

Ao ligar a renderização, houve uma perda de desempenho de 52.4% quando comparamos o algoritmo de *frustum culling* híbrido com o melhor obtido em CPU. Esta queda de desempenho pode ser explicada pelo monitoramento do algoritmo para determinar se os cálculos devem ser feitos em CPU ou em GPU. Quando o algoritmo é executado em GPU ocorre um gargalo por ter muitas tarefas a serem executadas na placa gráfica e retorno dos dados da GPU para CPU. A parte onde o *frustum culling* híbrido ganhou da melhor abordagem em CPU é explicada por ter poucas coisas para serem renderizadas

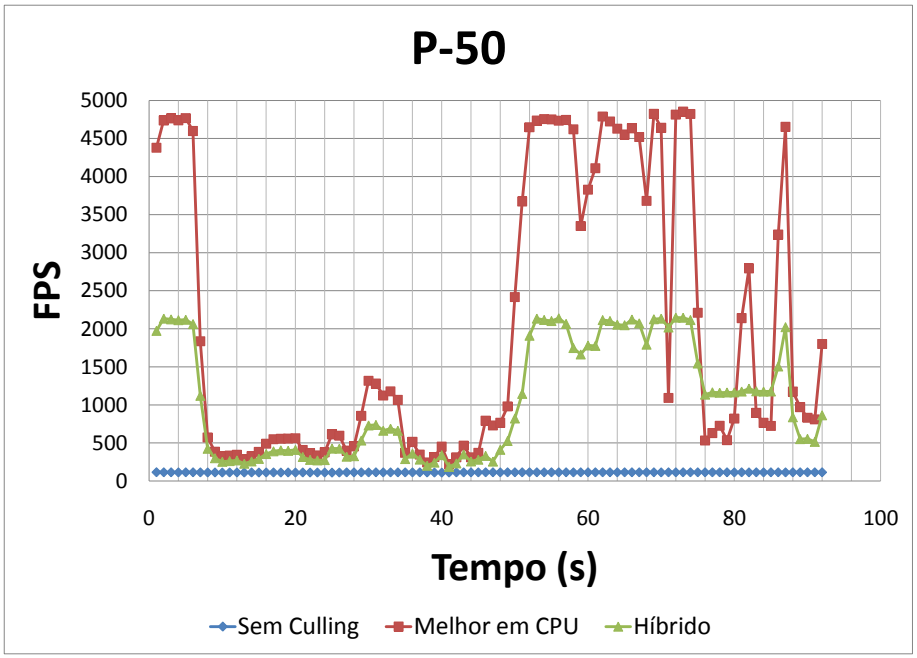

Figura 6.7: Resultados do frustum culling híbrido com render ligado.

e o algoritmo híbrido está rodando em GPU.

Mesmo não tendo o resultado esperado, o algoritmo de *frustum culling* híbrido talvez possa trazer ganhos no caso de várias placas gráficas, visto que ele não dividiria recursos com a renderização.