5
Comparação e Resultados

Neste capítulo, são apresentados e comparados os resultados obtidos com os algoritmos discutidos ao longo desta dissertação. Os testes foram realizados com as seguintes implementações:

- *VICP (CPU)*: VICP com estrutura de dados apenas na CPU e função de transferência pré-integrada;
- *VICP (GPU)*: VICP com estrutura de dados na placa gráfica, proposta na seção (4.1.1), e função de transferência pré-integrada;
- *Traçado de Raios (Pré-integração)*: Traçado de Raios, com função de transferência pré-integrada;
- *Traçado de Raios (Integração na GPU)*: Traçado de Raios com a integração de segmentos lineares na GPU (seção 4.2).

As duas implementações do algoritmo de Traçado de Raios utilizam a estrutura de dados em GPU descrita na seção (4.1.2).

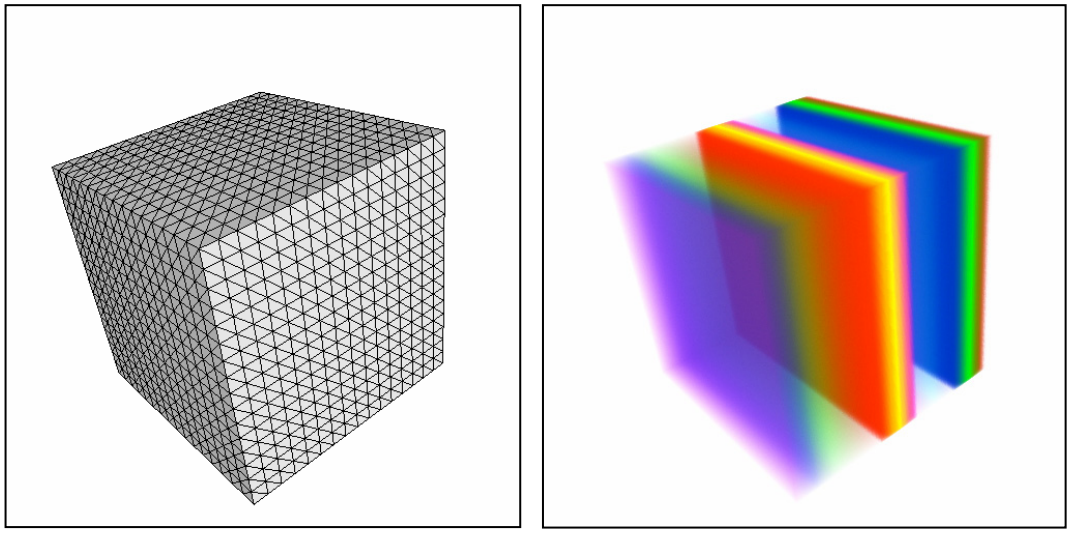
Além do Traçado de Raios, a integração de segmentos lineares da seção (4.2) foi aplicada também ao VICP. Entretanto, apesar do código fonte adicionado ao Traçado de Raios ser quase que diretamente mapeado para o VICP, o desempenho obtido foi insuficiente para permitir a visualização interativa, pois o programa por fragmento passou a apresentar um custo muito elevado. Assim, descartamos a utilização dessa técnica no VICP.

Os modelos utilizados para os testes são descritos na Tabela 6, enquanto as imagens geradas a partir desses modelos são apresentadas nas Figuras 29 a 33.

Tabela 6 – Características das malhas utilizadas para os testes de desempenho.

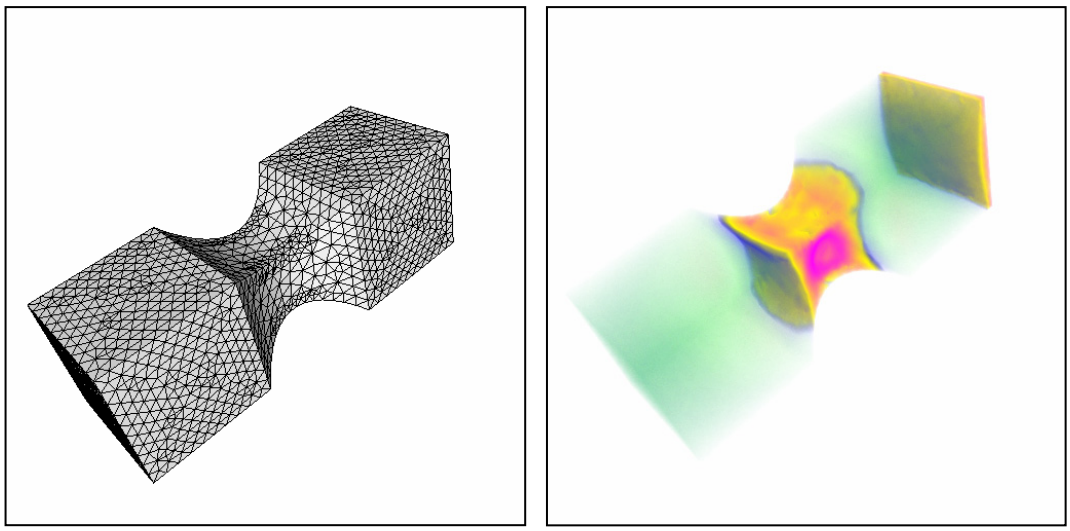
Malha	Num. de tetraedros	Num. de vértices	Propriedade visualizada
Grade 16x16x16	24.576	4.913	Escalar 1
Grade 32x32x32	196.608	35.937	Escalar 1
Barra Fixa	27.691	5.790	Tensão XX
Roda	31.725	6.855	Tensão XX
Bluntfin	224.874	40.960	Densidade
Oxygen	616.050	109.744	Momento X

As grades de 16x16x16 e 32x32x32 (Figura 29) são grades cartesianas, geradas sinteticamente e posteriormente decompostas em tetraedros, com uma propriedade escalar que varia em apenas uma direção. A malha *Barra Fixa* (Figura 30) é um modelo de elementos finitos de uma barra presa por uma das extremidades e com uma força aplicada na direção longitudinal. A *Roda* (Figura 31) também é uma malha de elementos finitos, à qual foi aplicada uma força ao longo de uma direção. *Bluntfin* (Figura 32) (Hung & Buning, 1990), que é disponibilizado pela *NASA Advanced Supercomputing Division*, é uma malha estruturada decomposta em tetraedros lineares. Essa malha apresenta uma curvatura em uma das extremidades e contém resultados da simulação do fluxo de um fluido na direção longitudinal. Finalmente, a malha *Oxygen* (ou *Liquid Oxygen Post*) (Rogers et al., 1986), também disponibilizado pela *NASA Advanced Supercomputing Division*, representa o fluxo de oxigênio líquido ao longo de um disco (Figura 33), com um "poste" cilíndrico posicionado perpendicularmente no centro do disco.



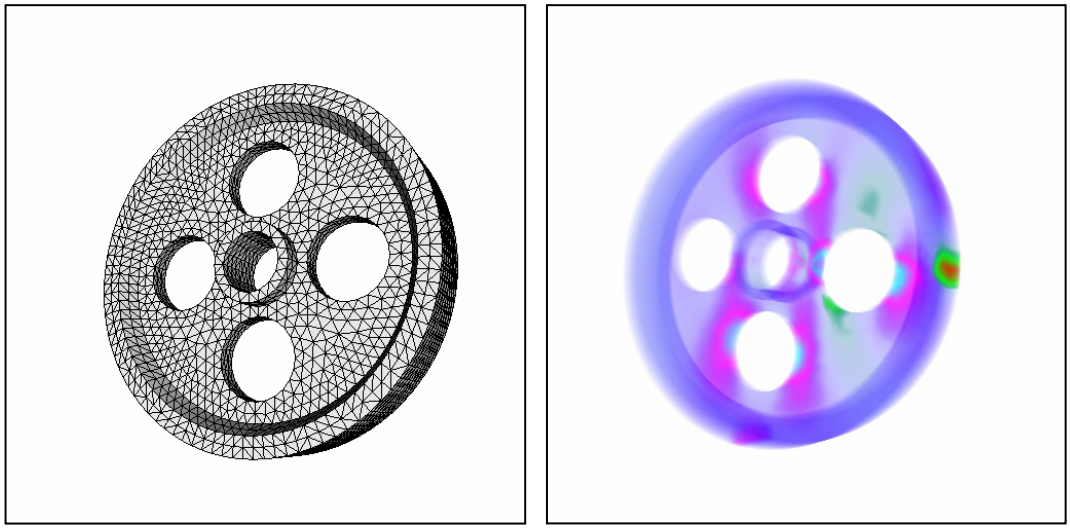
(a) **(b)**

Figura 29 – (a) Imagem da grade de 16x16x16. (b) Visualização volumétrica da grade de 16x16x16.



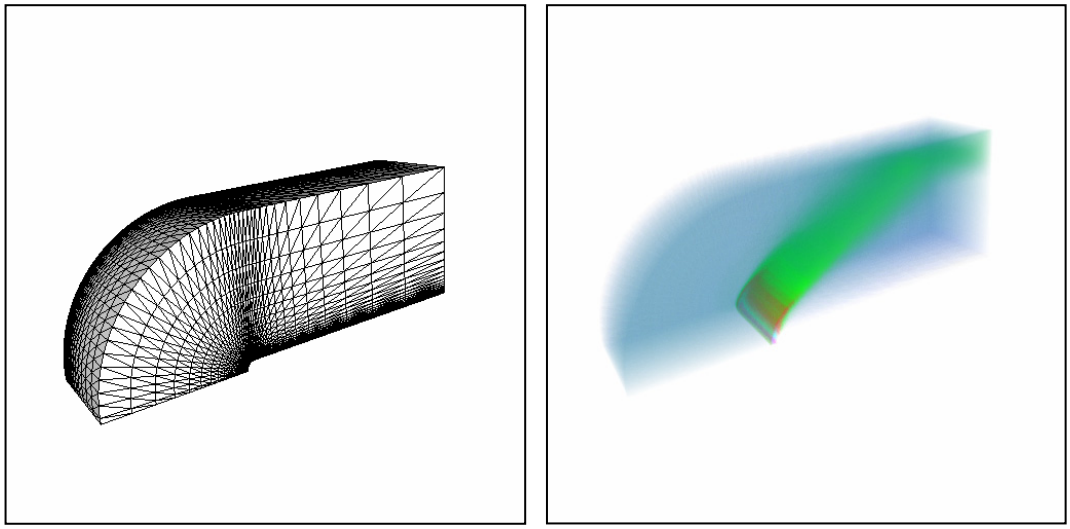
(a) **(b)**

Figura 30 – (a) Imagem da malha *Barra Fixa*. (b) Visualização volumétrica da malhas *Barra Fixa*.



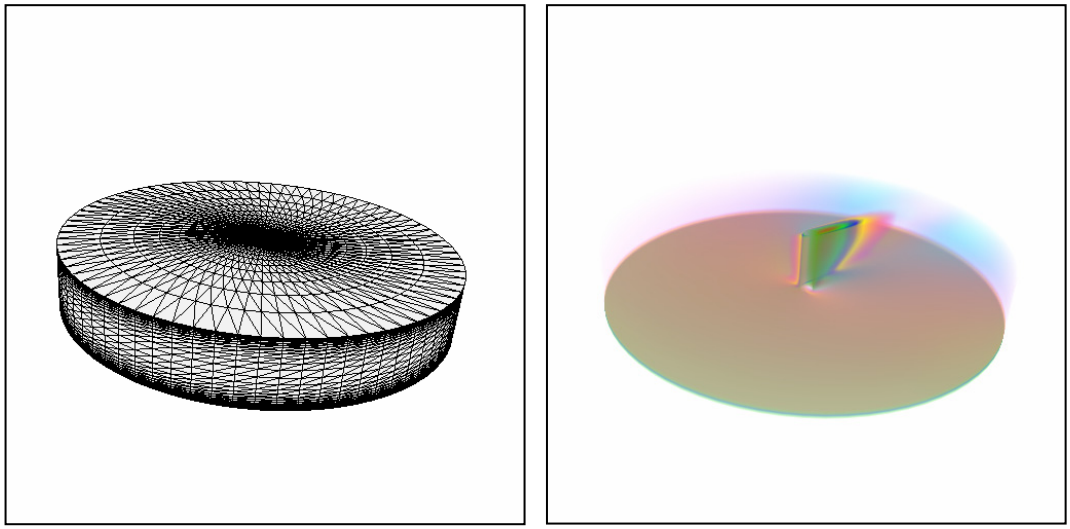
(a) **(b)**

Figura 31 – (a) Imagem da malha *Roda*. (b) Visualização volumétrica da malha *Roda*.



(a) **(b)**

Figura 32 – (a) Imagem da malha *Bluntfin*. (b) Visualização volumétrica da malha *Bluntfin*.



(a) **(b)**

Figura 33 – (a) Imagem da malha *Oxygen*. (b) Visualização volumétrica da malha *Oxygen*.

5.1. Desempenho

Os testes de desempenho foram executados em uma máquina com processador *Intel Pentium 4,* de *2,53GHz* e *512MB* de memória *RAM*. A placa gráfica utilizada foi uma *NVIDIA GeForce 6800 GT*, com *256MB* de memória e *AGP8X*, e os resultados foram medidos para uma janela de 512x512 pixels. As implementações dos programas por vértice e por fragmento foram realizadas utilizando a linguagem *Cg* (NVIDIA, 2004a), com os perfis de programação *vp40* e *fp40*.

Na Tabela 7, são apresentados os resultados para as duas implementações do VICP. Podemos observar que o desempenho do VICP (GPU) foi significativamente melhor que o VICP (CPU), mesmo sendo necessário aumentar o esforço do programa por fragmento com os acessos às texturas da estrutura de dados. Ao armazenarmos a malha na GPU, reduzimos drasticamente o custo da transferência de dados para a placa gráfica. Em nossa implementação, o novo gargalo passou a ser a rasterização. Isso foi comprovado observando-se que, conforme o tamanho da janela foi aumentado, o que implica em um número maior de fragmentos processados, houve queda de desempenho no VICP (GPU), principalmente para as malhas menores. No caso do VICP (CPU), os resultados se mantiveram estáveis. Para uma janela de 800x800, por exemplo, os valores mínimo e máximo de quadros por segundo obtidos para a Barra Fixa, com o VICP

(GPU), reduziram-se a 17,86 e 20,75, respectivamente, em comparação com os valores 24,63 e 26,67, como mostra a Tabela 7. Considerando a taxa atual de evolução da rasterização de fragmentos (Lefohn et al., 2004), acreditamos que, em pouco tempo, será possível obter resultados para janelas maiores equivalentes aos encontrados para a janela de 512x512 que foi utilizada.

Tabela 7 – Comparação dos resultados, em número de quadros por segundo (qd/s) e tetraedros por segundo (tet/s), dos algoritmos VICP (CPU) e VICP (GPU).

Malhas	VICP (CPU)				VICP (GPU)			
	Min. qd/s	Máx. qd/s	Min. tet/s	Máx. tet/s	Min. qd/s	Máx. qd/s	Min. tet/s	Máx. tet/s
Grade 16x16x16	11,85	12,55	291K	308K	14,53	21,32	357K	524K
Grade 32x32x32	1,43	1,51	281K	297K	3,42	3,74	672K	735K
Barra Fixa	9,84	10,67	272K	295K	24,63	26,67	682K	739K
Roda	8,76	9,28	278K	294K	22,08	22,88	700K	726K
Bluntfin	1,26	1,36	283K	306K	3,09	3,50	695K	787K
Oxygen	0,46	0,47	283K	290K	0,98	1,21	604K	745K

Uma desvantagem do VICP (e, em geral, dos algoritmos de projeção de células) é a necessidade de ordenação das células, considerando o modelo óptico de emissão e absorção. Neste trabalho, a ordenação foi realizada em CPU com a heurística MPVONC (Williams, 1992), que, mesmo sendo a mais eficiente extensão do MPVO reportada até o momento (Cook et al., 2004), impõe um limite quanto ao número de células que podem ser ordenadas por segundo. A Tabela 8 mostra os tempos necessários para ordenar algumas malhas de tamanhos variados. Considerando o tempo mínimo para a ordenação da grade 16x16x16, por exemplo, o número máximo de quadros desenhados por segundo é igual a 1 / 0,015 = 66,67, o que não representa um fator limitante para o algoritmo VICP (GPU), que consegue desenhar até 20 quadros por segundo. Entretanto, conforme o tamanho da malha aumenta, o desempenho do VICP (GPU) se aproxima do limite imposto pela ordenação. No caso da malha Oxygen, o número de quadros por segundo que o algoritmo consegue desenhar (1,21) é próximo ao máximo da ordenação (1 / 0,75 = 1,33), o que sugere que a ordenação passa a ser um forte fator limitante. Mesmo que fosse utilizado o algoritmo *Projected Tetrahedra*

(Shirley & Tuchman, 1990), mais eficiente por usar os recursos convencionais das placas gráficas, porém menos flexível, não seria possível transpor o limite imposto pela ordenação.

Tabela 8 – Tempos, em segundos, necessários à ordenação de visibilidade das células de diversas malhas.

Malhas	**Tempo (s)**	
	Min.	**Máx.**
Grade 16x16x16	0,015	0,031
Grade 32x32x32	0,23	0,27
Bluntfin	0,25	0,30
Oxygen	0,75	0,82

No Traçado de Raios, a ordenação de visibilidade é realizada de forma implícita, conforme cada raio é propagado ao longo da malha de tetraedros. Dessa forma, o Traçado de Raios não sofre com o limite imposto pela ordenação das células, como o VICP. Uma comparação entre o VICP (GPU) e o Traçado de Raios (Pré-integração) é apresentada na Tabela 9. Uma otimização possível para o Traçado de Raios consiste em interromper a propagação do raio quando a opacidade acumulada atinge um determinado valor. Porém, essa otimização não foi utilizada para a medição dos resultados da Tabela 9. O Traçado de Raios se mostrou competitivo com VICP, principalmente para as malhas maiores e quase convexas, como Bluntfin e Oxygen, nas quais o limite imposto pela ordenação das células começa a ser relevante. O gargalo do Traçado de Raios se localiza na rasterização, uma vez que quase todas as operações são realizadas no programa por fragmento. Assim, esse é um algoritmo que pode se beneficiar muito com a evolução das placas gráficas.

Uma desvantagem do Traçado de Raios está relacionada ao tratamento de malhas não-convexas. A “descamação” das malhas requer que, para cada camada, sejam desenhadas todas as faces externas. Conseqüentemente, em malhas com muitas concavidades, isso pode implicar em uma significativa perda de desempenho. Porém, para a maioria das malhas reais isso não deve representar um problema.

Tabela 9 – Comparação dos resultados, em número de quadros por segundo (qd/s) e tetraedros por segundo (tet/s), do VICP (GPU) e o Traçado de Raios (Pré-integração), para diversas malhas.

Malhas	VICP (GPU)				Traçado de Raios (Pré-integração)			
	Min. qd/s	Máx. qd/s	Min. tet/s	Máx. tet/s	Min. qd/s	Máx. qd/s	Min. tet/s	Máx. tet/s
Grade 16x16x16	14,53	21,32	357K	524K	8,53	11,63	210K	286K
Grade 32x32x32	3,42	3,74	672K	735K	3,85	5,98	757K	1,18M
Barra Fixa	24,63	26,67	682K	739K	8,76	14,88	243K	412K
Roda	22,08	22,88	700K	726K	7,80	14,53	247K	461K
Bluntfin	3,09	3,50	695K	787K	6,09	8,76	1,37M	1,97M
Oxygen	0,98	1,21	604K	745K	3,40	5,91	2,09M	3,64M

Os resultados dos testes para o Traçado de Raios (Integração na GPU) são comparados na Tabela 10. Foram utilizadas funções de transferência com 10 e 255 segmentos igualmente espaçados em função do campo escalar. Pela tabela, podemos notar que o desempenho desta abordagem foi inferior ao Traçado de Raios (Pré-integração). Isto pode ser explicado pelo aumento no número de passos necessários para a propagação de um raio, que, neste caso, é proporcional ao número de iso-superfícies atravessadas. Quanto maior o número de segmentos lineares da função de transferência, maior será o número de iso-superfícies em um tetraedro e mais segmentos devem ser integrados ao longo do raio. O número de iso-superfícies também depende da variação do campo escalar no interior de cada tetraedro. Entretanto, para malhas de elementos finitos, a tendência é não haver uma grande variação do campo escalar em um tetraedro, uma vez que grandes variações indicam problemas de discretização das malhas.

Embora a diferença de desempenho do Traçado de Raios (Integração na GPU) com 255 segmentos em relação ao Traçado de Raios (Pré-integração) seja significativa, no caso de 10 segmentos a consideramos aceitável. Funções de transferência definidas por poucos segmentos lineares também podem ser muito úteis. Para a visualização de elementos finitos é comum a criação de escalas de cores a partir de poucas cores básicas (2 ou 3, por exemplo), que são interpoladas linearmente. Outros pontos de controle podem ser definidos pelo usuário para

controlar a opacidade dos dados que se deseja visualizar, mas é comum termos poucos segmentos definindo a função de transferência. É importante notar que a integração de segmentos lineares na GPU ainda tem a vantagem de oferecer uma qualidade de imagem superior (seção 5.3) à pré-integração e de não ser restrita a funções de transferência unidimensionais.

Tabela 10 – Comparação dos resultados, em número de quadros por segundo (qd/s) e tetraedros por segundo (tet/s), para o Traçado de Raios (Integração na GPU), com 10 e 255 segmentos, para diversas malhas.

Malhas	Traçado de Raios (Integração na GPU) 10 seg.				Traçado de Raios (Integração na GPU) 255 seg.			
	Min. qd/s	Máx. qd/s	Min. tet/s	Máx. tet/s	Min. qd/s	Máx. qd/s	Min. tet/s	Máx. tet/s
Grade16x16x16	6,15	9,14	151K	225K	2,02	5,76	50K	142K
Grade32x32x32	3,00	5,08	590K	999K	1,71	3,30	336K	649K
Barra Fixa	6,27	11,22	174K	311K	2,41	4,81	67K	133K
Roda	7,35	10,67	233K	339K	3,86	7,19	122K	228K
Bluntfin	3,50	7,11	787K	1,60M	2,98	5,04	670K	1,13M
Oxygen	2,32	4,21	1,43M	2,59M	1,22	2,38	752K	1,47M

5.2. Memória

Para o VICP, não é necessário armazenar dados de uma malha de tetraedros na placa gráfica. Entretanto, utilizando a estrutura de dados em GPU proposta na seção (4.1.1), que ocupa 119 *bytes* por tetraedro, foi possível melhorar o desempenho desse algoritmo de forma significativa. Embora esta seja uma alternativa atraente, ela também apresenta algumas possíveis desvantagens. No VICP (GPU), não podemos descartar a estrutura de dados na CPU, uma vez que ela é necessária para a ordenação das células. Assim, os dados são armazenados de forma redundante. Quando apenas as estruturas de dados armazenadas em CPU são utilizadas, também podem ser implementados algoritmos que alteram a malha dinamicamente, como alguns algoritmos de multi-resolução (*LOD*). No caso de

uma estrutura de dados armazenada na placa gráfica, como a da seção (4.1.1), isso pode se tornar mais difícil, pois as respectivas texturas devem ser atualizadas.

O Traçado de Raios requer uma estrutura de dados armazenada na placa gráfica, como a apresentada na seção (4.1.2), que ocupa 96 *bytes* por tetraedro. Na CPU, pode-se armazenar apenas a lista das faces externas da malha, com alguns atributos adicionais. Dessa forma, uma vantagem do Traçado de Raios é que a replicação de dados é muito menor do que no caso do VICP (GPU).

Mesmo com a estrutura de dados e o controle da renderização quase que completamente na GPU, ainda é possível implementar facilmente algoritmos como planos de corte e visualização de iso-superfícies diretamente na placa gráfica, o que representa um grande potencial dessa abordagem para diversas aplicações. Um plano de corte, por exemplo, pode ser implementado efetivamente "interrompendo" o raio, caso este intercepte o plano ainda no interior do tetraedro.

Além das estruturas de dados, outras informações, como a função de transferência pré-integrada, também ocupam espaço na memória da placa gráfica. Uma textura 3D de 4 componentes (R,G,B,A), com 8 *bits* por componente e dimensões 128x128x128 (necessária para uma boa qualidade da imagem final), ocupa 8.388.608 *bytes*. Se considerarmos a estrutura de dados proposta para o VICP (GPU) na seção (4.1.1), o espaço ocupado equivale a uma malha de $8.388.608/119 \approx 70.493$ tetraedros. Para a estrutura de dados utilizada no Traçado de Raios (seção 4.1.2), esse espaço equivale a $8.388.608/96 \approx 87.381$ tetraedros. Assim, embora a memória ocupada pela função de transferência pré-integrada não represente em si um problema de armazenamento para as placas gráficas modernas, deve-se notar que a memória da placa gráfica pode ser um fator limitante para a visualização de modelos muito grandes, quando utilizada a estruturação de dados na GPU.

5.3. Qualidade de imagem

O VICP (CPU) e o VICP (GPU) deveriam apresentar a mesma qualidade de imagem que o Traçado de Raios (Pré-integração), considerando-se uma mesma função de transferência. No entanto, o Traçado de Raios gerou imagens melhores, principalmente para malhas maiores, como ilustrado na Figura 34. A diferença

pode ser explicada pela precisão utilizada para a composição de cores. No caso do VICP, a contribuição de cada tetraedro foi composta em uma tela de projeção com formato de imagem convencional, de 8 *bits* por componente, uma vez que o driver da placa de vídeo utilizada ainda não suporta completamente formatos mais precisos. Por outro lado, no Traçado de Raios, as cores foram compostas com a precisão interna da placa gráfica (*floats* de 32 *bits* ou *halfs* de 16 *bits*).

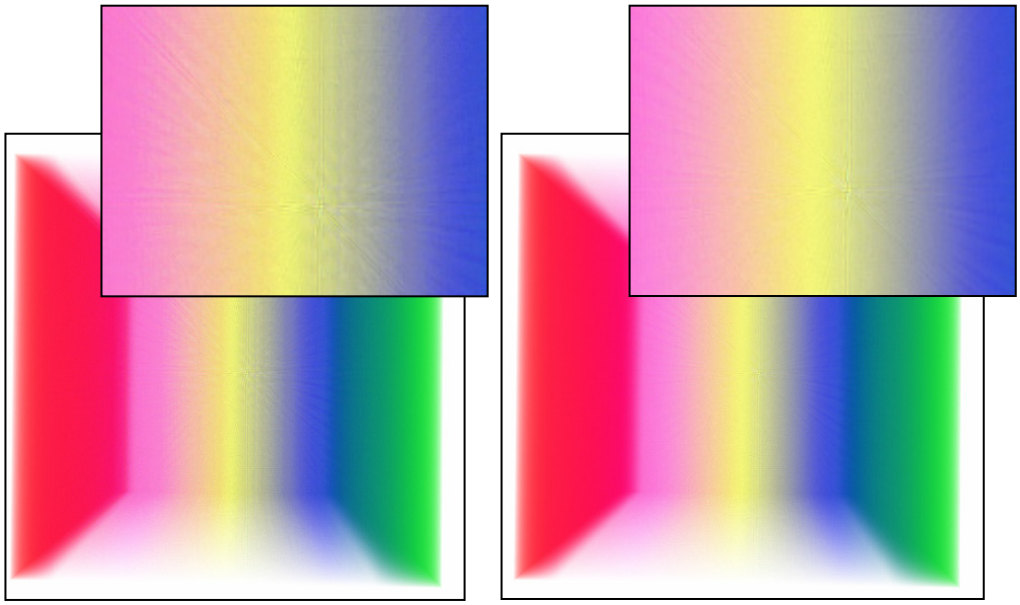
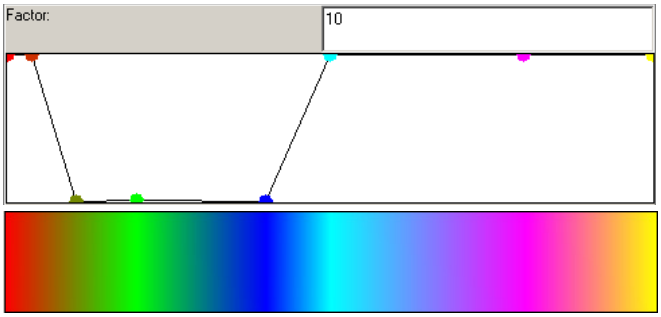
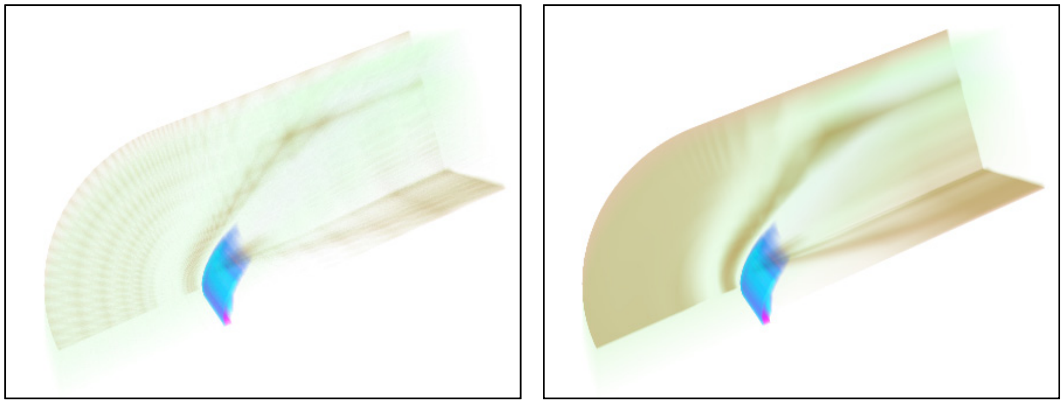


Figura 34 – Diferença entre a qualidade da imagem gerada pelo VICP (a) e Traçado de Raios (Pré-integração) (b), para grade 32x32x32, com a mesma função de transferência, armazenada em uma textura 3D de 128x128x128.

O Traçado de Raios (Integração na GPU) gerou imagens com maior precisão e qualidade superior ao Traçado de Raios (Pré-integração). Em muitas situações, como a ilustrada na Figura 35, a diferença entre as imagens torna-se significativa. Neste caso, a malha Bluntfin, cujo tamanho das células varia muito, apresenta falhas visuais que podem ser notadas em algumas situações. A diferença pode ser atribuída à amostragem finita da função de transferência pré-integrada. Como as coordenadas de textura variam no intervalo [0,1), a distância que o raio percorre em um tetraedro é normalizada em função do comprimento da maior aresta da malha. Tetraedros muito pequenos passam a corresponder a posições muito próximas na textura 3D, enquanto que os maiores correspondem a posições distantes.



(a)



(b) (c)

Figura 35 – Diferença entre a qualidade da imagem do Traçado de Raios (Pré-integração) e Traçado de Raios (Integração na GPU): (a) função de transferência utilizada; (b) Traçado de Raios (Pré-integração), com a função de transferência pré-integrada e armazenada em uma textura 3D de 128x128x128; (c) Traçado de Raios (Integração na GPU).

No Traçado de Raios também é importante manter o parâmetro da equação de cada raio atualizado a cada passo do algoritmo, com a maior precisão possível. Ao compactarmos esse parâmetro em um tipo *half*, de 16 *bits*, a perda de precisão resultou no problema de amostragem conhecido como *aliasing*, para alguns casos do algoritmo de Traçado de Raios. Esse problema ocorreu para malhas maiores, como Bluntfin e Oxygen, como ilustrado na Figura 36. Com a precisão de 32 *bits*, não observamos nenhuma falha visual relativa ao fenômeno de *aliasing*. Entretanto, esse é um problema que poderá ocorrer em malhas muito maiores do que as que foram visualizadas.

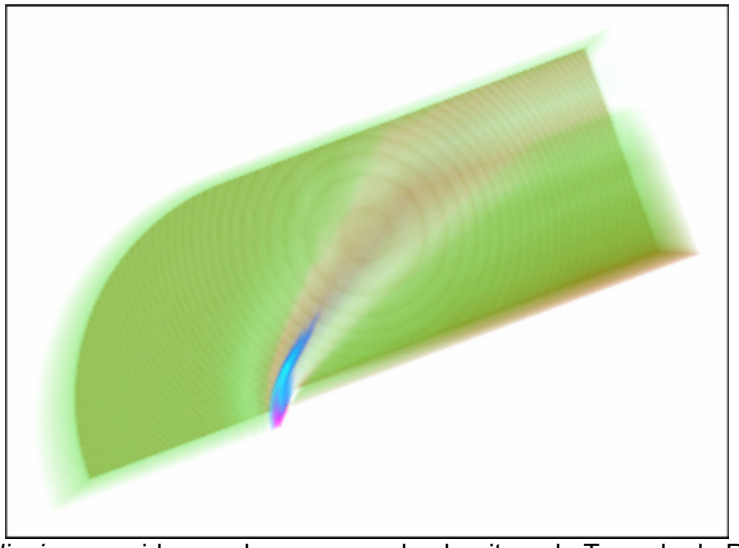


Figura 36 – *Aliasing* ocorrido em alguns casos do algoritmo de Traçado de Raios, para a malha Bluntfin, quando a precisão de 16 *bits* é utilizada para o parâmetro da equação do raio.

5.4. Modificação interativa da função de transferência

Um inconveniente da pré-integração está no tempo necessário para a integração da função de transferência. Quanto maior a textura 3D, maior o tempo de integração e atualização dessa função. Na literatura são encontradas algumas técnicas aproximativas (Roettger & Ertl, 2002; Weiler et al., 2003a) que aceleram dramaticamente este processo. Entretanto, cada uma apresenta vantagens e desvantagens. Entre as possíveis desvantagens, encontra-se a perda de qualidade de imagem para algumas funções de transferência, devido a erros de aproximação. Mas, em geral, apresentam resultados satisfatórios.

Neste trabalho, realizamos a pré-integração da função de transferência com o auxílio da placa gráfica, como proposto por (Roettger & Ertl, 2002). Para isso, foram desenhados n_l retângulos de dimensões n_{sf} e n_{sb}, sendo n_l a dimensão ao longo do comprimento l de um raio que atravessa um tetraedro, e n_{sf} e n_{sb} as dimensões dos valores dos escalares de entrada e saída do raio no tetraedro. Para a integração de cada valor em função de (s_f, s_b, l) foi utilizado o método de integração de segmentos lineares na GPU proposto na seção (4.2). Os resultados de cada retângulo desenhado foram armazenados em uma textura 3D. Com uma textura de dimensões 128x128x128 e 32 segmentos lineares, a pré-integração pôde ser realizada em aproximadamente 1,3s.

Para o algoritmo de Traçado de Raios (Integração na GPU), a atualização da função de transferência é quase instantânea, pois o custo da pré-integração não existe, o que permite a realização de modificações interativas.