Pontifícia Universidade Católica
do Rio de Janeiro

Leonardo Quatrin Campagnolo

Visualização volumétrica precisa baseada em integração numérica adaptativa

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática da PUC-Rio.

Orientador: Prof. Waldemar Celes Filho

Rio de Janeiro
Agosto de 2015

Pontifícia Universidade Católica
do Rio de Janeiro

Leonardo Quatrin Campagnolo

Visualização volumétrica precisa baseada em integração numérica adaptativa

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Waldemar Celes Filho
Orientador
Departamento de Informática — PUC-Rio

Prof. Hélio Côrtes Vieira Lopes
Departamento de Informática — PUC-Rio

Prof. Ricardo Marroquim
Universidade Federal do Rio de Janeiro (UFRJ)

Prof. José Eugênio Leal
Coordenador Setorial do Centro Técnico Científico — PUC-Rio

Rio de Janeiro, 19 de agosto de 2015

Leonardo Quatrin Campagnolo

Bacharel em Ciência da Computação, graduou-se em 2012 pela Universidade Federal de Santa Maria. Possui interesse principalmente em Processamento Gráfico e Visualização Científica.

Ficha Catalográfica

Campagnolo, Leonardo Quatrin

Visualização volumétrica precisa baseada em integração numérica adaptativa / Leonardo Quatrin Campagnolo; orientador: Waldemar Celes Filho. — Rio de Janeiro : PUC-Rio, Departamento de Informática, 2015.

76 f: il. (color.); 29,7 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2015.

Inclui bibliografia.

1. Informática – Teses. 2. Visualização volumétrica;. 3. Integração adaptativa;. 4. Controle de erro;. 5. Regra de Simpson.. I. Filho, Waldemar Celes. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Agradecimentos

Primeiramente, gostaria de agradecer aos meus pais, Antonio e Liséte, pela vida e pelo esforço deles para que eu pudesse realizar toda esta trajetória. Agradecer por cada lição que me foi ensinada durante todos esses anos, me tornando uma pessoa melhor.

Um agradecimento especial para minha esposa, Sabrina, pelo apoio, pelos desabafos e por todo o carinho transmitido durante este processo. Pela paciência nos momentos em que eu precisava estar focado no trabalho e por sempre estar ao meu lado nos bons e maus momentos.

Ao meu orientador, Prof. Dr. Waldemar Celes Filho, por toda a dedicação e por ter me guiado neste trabalho. Pelo imenso aprendizado que obtive com ele e por ser um inigualável exemplo de professor.

Queria também agradecer ao meu orientador de graduação, Prof. Dr. Cesar Tadeu Pozzer, por ter me incentivado a realizar meu mestrado na PUC-Rio e por todo o ensinamento que me foi passado. Também ao Eduardo Ceretta Dalla Favera por ter me ajudado nos meus primeiros meses de adaptação no Rio de Janeiro.

Gostaria de agradecer aos amigos e colegas de trabalho, por todas as conversas que tornaram este processo mais tranquilo. Não achei justo citar os nomes de cada um pois provavelmente acabaria me esquecendo de alguém importante.

Gostaria também de agradecer ao CNPq, Conselho Nacional de Desenvolvimento Científico e Tecnológico, pelo apoio financeiro durante esta etapa.

Por fim, gostaria de agradecer a Deus por cada dia e por cada momento de calma e tranquilidade para que eu pudesse concluir esta etapa.

Resumo

Campagnolo, Leonardo Quatrin; Filho, Waldemar Celes. **Visualização volumétrica precisa baseada em integração numérica adaptativa**. Rio de Janeiro, 2015. 76p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Um dos principais desafios em algoritmos de visualização volumétrica é calcular a integral volumétrica de maneira eficiente, mantendo uma precisão mínima adequada. Geralmente, métodos de integração numérica utilizam passos de tamanho constante, não incluindo nenhuma estratégia de controle numérico. Como uma possível solução, métodos numéricos adaptativos podem ser utilizados, pois conseguem adaptar o tamanho do passo de integração dada uma tolerância de erro pré-definida. Em CPU, os algoritmos adaptativos de integração numérica são, normalmente, implementados recursivamente. Já em GPU, é desejável eliminar implementações recursivas. O presente trabalho propõe um algoritmo adaptativo e iterativo para a avaliação da integral volumétrica em malhas regulares, apresentando soluções para manter o controle do passo da integral interna e externa. Os resultados do trabalho buscaram comparar a precisão e eficiência do método proposto com o modelo de integração com passo de tamanho constante, utilizando a soma de Riemann. Verificou-se que o algoritmo proposto gerou resultados precisos, com desempenho competitivo. As comparações foram feitas em CPU e GPU.

Palavras–chave

Visualização volumétrica; Integração adaptativa; Controle de erro; Regra de Simpson.

Abstract

Campagnolo, Leonardo Quatrin; Filho, Waldemar Celes (Advisor). **Accurate Volume Rendering based on Adaptive Numerical Integration**. Rio de Janeiro, 2015. 76p. MSc Thesis — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

One of the main challenges in volume rendering algorithms is how to compute the Volume Rendering Integral accurately, while maintaining good performance. Commonly, numerical methods use equidistant samples to approximate the integral and do not include any error estimation strategy to control accuracy. As a solution, adaptive numerical methods can be used, because they can adapt the step size of the integration according to an estimated numerical error. On CPU, adaptive integration algorithms are usually implemented recursively. On GPU, however, it is desirable to eliminate recursive algorithms. In this work, an adaptive and iterative integration strategy is presented to evaluate the volume rendering integral for regular volumes, maintaining the control of the step size for both internal and external integrals. A set of computational experiments were made comparing both accuracy and efficiency against the Riemann summation with uniform step size. The proposed algorithm generates accurate results, with competitive performance. The comparisons were made using both CPU and GPU implementations.

Keywords

Volume rendering; Adaptive integration; Error control; Simpson's rule.

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

A visualização volumétrica é um método utilizado para extrair informações de dados volumétricos através de técnicas de computação gráfica e processamento de imagens, e preocupa-se com a representação, manipulação e renderização dessas informações. Dados volumétricos são estruturas tridimensionais que podem possuir ou não alguma informação dentro delas, podendo ou não consistir em superfícies e arestas tangíveis (concretas) ou podendo ser muito volumosas para serem representadas geometricamente [6].

A visualização volumétrica atua em áreas como: biomedicina, geofísica, dinâmica de fluidos, modelos finitos, bioquímica, simulações numéricas e imagens médicas, auxiliando na percepção e entendimento dos dados. Até onde se sabe, Drebin et al. [7] propuseram pela primeira vez um algoritmo de visualização volumétrica, a partir do desenho de um conjunto de imagens médicas contendo diferentes materiais.

Algoritmos de visualização volumétrica consistem em avaliar, classificar e mapear um campo tridimensional para um plano de visualização (2D). As duas principais abordagens conhecidas baseiam-se em qual porção do volume é avaliado. São chamadas de Visualização de Superfícies e Visualização Volumétrica.

A técnica de Visualização de Superfícies [8], também conhecida por Visualização Volumétrica Indireta, é baseada em algoritmos que buscam extrair e gerar a geometria das iso-superfícies do volume. A renderização da geometria construída para o plano de visualização é feita utilizando algoritmos convencionais de desenho de polígonos.

Já a técnica de Visualização Volumétrica, também conhecida por Visualização Volumétrica Direta, compreende técnicas onde todo o volume é avaliado e utilizado para gerar a visualização.

Os algoritmos baseados em Visualização Volumétrica acabam sendo mais complexos devido à necessidade de avaliar todo o volume e calcular a contribuição de cada parte do mesmo, diferentemente do algoritmo de visualização de superfícies. De fato, é possível capturar uma quantidade maior de informações do volume, coisa que o algoritmo de superfícies não proporciona. Uma das vantagens da Visualização Direta em relação à Visualização Indireta é a flexibilidade em determinar como cada parte do volume irá constribuir para a imagem final [9], através do uso de funções de transferência.

Devido às técnicas de visualização volumétrica tomarem um alto tempo de processamento para serem executadas, somado aos tamanhos dos conjuntos de dados sendo visualizados, é comum utilizar-se de programação em placa gráfica para acelerar o processo de avaliação. Essa é uma prática muito comum devido à facilidade de paralelização da maioria dos algoritmos de visualização volumétrica.

O algoritmo de lançamento de raios para visualização volumétrica consiste em gerar a projeção do volume através da integração dos efeitos das propriedades óticas ao longo de cada raio lançado a partir da câmera do observador [1]. O modelo de iluminação, formulado pelas propriedades óticas somadas ao longo de cada raio lançado, resulta na conhecida integral volumétrica (VRI - *Volume Rendering Integral*)[1]. Um dos grandes desafios das técnicas de visualização volumétrica consiste em calcular a integral volumétrica de maneira eficiente, mantendo uma precisão mínima adequada.

Normalmente, a integral volumétrica é aproximada a partir da utilização de métodos de integração numérica com amostras equidistantes. A implementação se torna simples e intuitiva, principalmente em GPU. Porém, não há nenhum controle do erro numérico da integração. Dessa forma, se for adotado um passo muito grande, o processo será eficiente, porém não necessariamente preciso. Por outro lado, se for adotado um passo muito pequeno, a avaliação será mais precisa, porém a performance é geralmente comprometida.

Diante desse cenário, métodos de integração numérica adaptativos podem ser utilizados como uma solução alternativa. Nesses casos, o passo é adaptado de acordo com o erro numérico avaliado. Essa propriedade permite um melhor controle de erro e torna possível o uso de estratégias que permitem balancear a precisão e a performance da avaliação de maneira automática. Entretanto, calcular a integral volumétrica de forma adaptativa não é trivial. Foram identificados dois desafios principais:

- A integral volumétrica é composta por duas integrais (interna e externa), sendo que a interna calcula a transparência do meio e a externa calcula a contribuição final (RGBA). Como aplicar passos de integração adaptativos para ambas as integrais de maneira eficiente?
- Geralmente, algoritmos adaptativos são recursivos: como mapear uma solução apropriada para placas gráficas?

O presente trabalho apresenta uma estratégia de avaliação da integral volumétrica de maneira adaptativa para malhas regulares. Será feita a apre-

[1] descrita na Seção 2.4

sentação de um algoritmo iterativo, buscando contornar o caráter recursivo, normalmente apresentado nos algoritmos adaptativos de integração numérica.

Para que seja feita uma proposta de implementação coerente, sabendo que a integral volumétrica contém duas integrais que devem ser calculadas de maneira adaptativa, deve-se gerar uma forma de integrar e controlar o passo de integração das duas partes de maneira coerente. Duas abordagens foram propostas, variando a maneira de controlar o passo da integral externa.

Alguns resultados foram realizados, com diferentes modelos, para analisar a precisão e performance dos métodos implementados. Será demonstrado que o método desenvolvido neste trabalho consegue manter o controle numérico, preservando uma performance adequada.

Este texto está organizado da seguinte forma: No Capítulo 2 são apresentados alguns conceitos básicos relacionados à visualização volumétrica. No Capítulo 3 são apresentados alguns trabalhos relacionados. O Capítulo 4 apresenta a metodologia utilizada no trabalho. No Capítulo 5 são apresentados os resultados do trabalho e o Capítulo 6 contém algumas conclusões do trabalho, além de propostas de trabalhos futuros.

2
Conceitos básicos

Neste capítulo, serão apresentados alguns conceitos básicos dentro da área de visualização volumétrica necessários para o entendimento do método implementado neste trabalho. Será vista a integral volumétrica, sua discretização e também uma introdução sobre malhas estruturadas e não-estruturadas. Ao final, será apresentado o algoritmo de lançamento de raios, utilizado como base para o método de visualização volumétrica, e funções de transferência.

2.1
Regra de Simpson e Regra de Simpson Adaptativa

A regra de Simpson (2.1) consiste em aproximar uma função f em um intervalo de largura h, sendo $f(x)$ a avaliação da função real em um ponto x:

$$\int_t^{t+h} f(x)dx \approx S(t,h) = \frac{h}{6}\left[f(t) + 4f\left(t+\frac{h}{2}\right) + f(t+h)\right] \tag{2.1}$$

O método de Simpson adaptativo (*Adaptive Simpson's rule*), proposto por Kuncir [10] e McKeeman [11] e analisado by Lyness [12], busca subdividir recursivamente um intervalo de integração até que o erro aproximado $E(t,h)$ seja menor ou igual que um valor de erro pré-definido ϵ, passado por parâmetro.

Para aproximar um intervalo utilizando o método de Simpson adaptativo, de acordo com a equação (2.2), $D(t,h)$ (2.3) é somado a uma contribuição $E(t,h)$. Esse valor é calculado através da diferença entre a avaliação de todo o intervalo utilizando um passo de avaliação (h) e duas avaliações com passos de avaliação $\left(\frac{h}{2}\right)$, dividindo o intervalo em duas metades. O valor de $E(t,h)$ é calculado pela equação (2.4).

$$\int_t^{t+h} f(x)dx \approx D(t,h) + E(t,h) \tag{2.2}$$

$$D(t,h) = S\left(t,\frac{h}{2}\right) + S\left(t+\frac{h}{2},\frac{h}{2}\right) \tag{2.3}$$

$$E(t,h) = \frac{1}{15}\left|D(t,h) - S(t,h)\right| \tag{2.4}$$

A avaliação de erro estimado $E(t,h)$ é principalmente utilizada para determinar o critério de parada do algoritmo adaptativo (sugerido por Lyness [12]). Dessa forma, se o erro for maior que uma certa tolerância pré-definida, o intervalo é dividido em duas partes e os dois intervalos ($D(t,\frac{h}{2})$ e $D(t+\frac{h}{2},\frac{h}{2})$)

são recursivamente reavaliados até que o erro total estimado seja menor que a tolerância.

O Algoritmo 1 ilustra o pseudo-código do processo principal simplificado do algoritmo recursivo de Simpson adaptativo. Percebe-se que o valor de tolerância também é dividido pela metade, assim como o tamanho do passo. Isso garante que o erro total estimado irá respeitar a tolerância originalmente especificada para todo o intervalo. O pseudo-código do algoritmo de Simpson adaptativo (função principal e auxiliar) pode ser visto no Apêndice A.

Algorithm 1 Processo recursivo da regra de Simpson adaptativo.

Input: t, h, ϵ
Output: $\int(t, h, \epsilon)$
1: **if** $E(t,h) \leq \epsilon$ **then**
2: **return** $D(t,h) + E(t,h)$
3: **else**
4: **return** $\int(t, \frac{h}{2}, \frac{\epsilon}{2}) + \int(t + \frac{h}{2}, \frac{h}{2}, \frac{\epsilon}{2})$
5: **end if**

2.2 Malha estruturada

Um dado volumétrico normalmente consiste em um conjunto de elementos, contendo algum campo escalar ou vetorial, representando alguma propriedade. Cada um desses elementos normalmente possui uma localização 3D, podendo possuir ou não uma correlação espacial entre seus vizinhos e todo o volume em si. Para visualizar corretamente um dado volumétrico, é necessário ter uma estrutura que possibilite acessar os seus respectivos elementos de maneira coerente.

Normalmente, essas estruturas são chamadas de malhas estruturadas e não-estruturadas. Nas malhas estruturadas, cada elemento pode ser acessado através de uma indexação lógica espacial (i, j, k). A Figura 2.1 ilustra o caso mais simples de malha estruturada e a Figura 2.2 demonstra um tipo de malha estruturada com sua representação geométrica e sua representação lógica.

As malhas estruturadas podem ser interpretadas como um conjunto de duas formas principais, denominadas voxel e célula. As duas se diferenciam no modo em que são avaliadas. As duas estruturas são ilustradas pela Figura 2.3.

No caso dos voxels, cada um deles é definido como um ponto de malha no espaço 3D que possui uma região de influência com valor constante. Essa região possui um formato de hexaedro que não possui variação em torno do seu ponto central de malha (assim como a região formada na Figura 2.3a pelo ponto laranja). Em alguns algoritmos, a contribuição do voxel diminui à

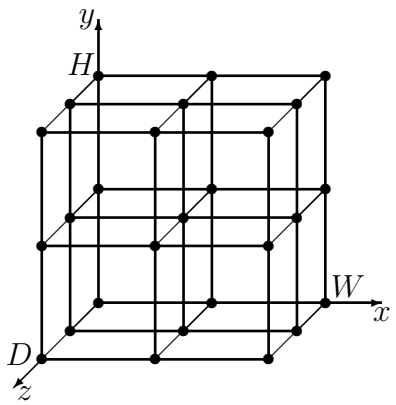


Figura 2.1 – Exemplo de malha estruturada.

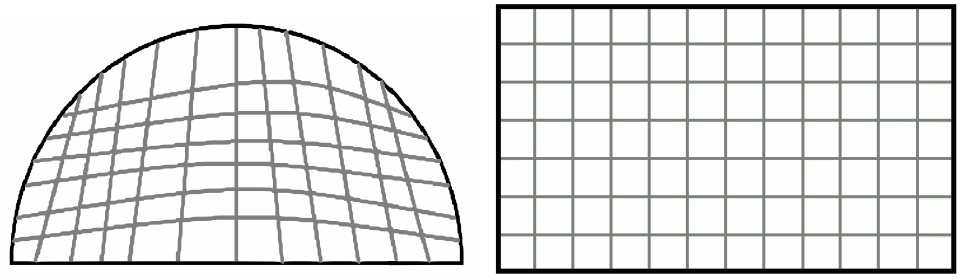


Figura 2.2 – Exemplo de malha estruturada com sua representação geométrica e lógica.

medida em que se distancia do ponto central. Em outros casos, o voxel tem uma contribuição constante em toda a região de influência [13].

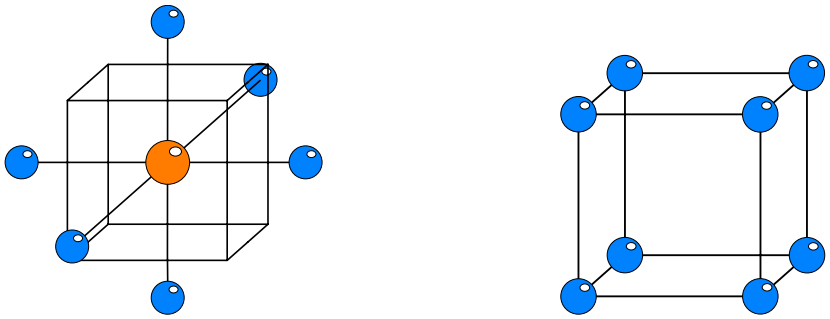
Já as células em um volume são normalmente interpretadas como um conjunto de hexaedros cujos vértices são pontos de malha e seu respectivo campo escalar é variável. Estima-se o campo escalar dentro de cada célula a partir da interpolação dos valores determinados nos seus vértices [13].

2.3 Malha não-estruturada

Para as malhas não-estruturadas, diferentemente das estruturadas, é necessário determinar explicitamente a correlação entre os dados, indicando os vizinhos de cada elemento da malha. A visualização volumétrica de malhas não-estruturadas foi inicialmente proposta por Garrity [14], utilizando o algoritmo de lançamento de raios.

As malhas não-estruturadas conseguem representar dados mais complexos, porém é necessário um espaço maior de armazenamento e o acesso é mais custoso. A Figura 2.4 ilustra um exemplo de malha não-estruturada.

Quando se avalia uma malha não-estruturada, sempre há a necessidade de armazenar os vértices de cada elemento (hexaedro, tetraedro, etc.) que a compõe. Na maioria dos casos, os elementos podem ser representados de duas



(a) voxel (b) célula

Figura 2.3 – Formas principais de representação das malhas estruturadas.

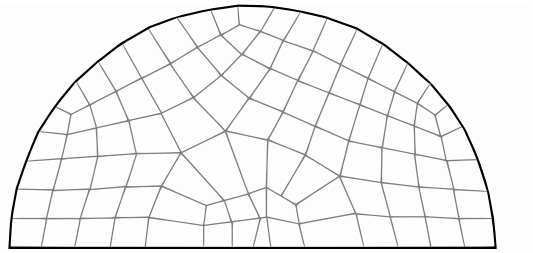


Figura 2.4 – Exemplo de malha não-estruturada.

principais formas: ou considera-se que o campo escalar no interior de cada elemento é constante, ou é definido um valor nos vértices de cada elemento e o campo escalar no seu interior é gerado a partir da interpolação de cada um desses valores.

Neste trabalho foram testados apenas modelos compostos por malhas estruturadas representadas pela Figura 2.1, podendo a proposta ser estendida para casos mais genéricos e para malhas não-estruturadas.

2.4 Integral volumétrica

O elemento fundamental da visualização volumétrica é a integral volumétrica [15]. Ela representa o modo como a luz interage com o modelo que está sendo visualizado. Normalmente, a construção da integral volumétrica baseia-se em um modelo ótico, que determina as características dessa interação da luz com o modelo e com o ambiente de visualização, que pode ser composto por uma ou mais fontes de luz. Geralmente, é utilizado o modelo de *low-albedo*[1], onde as partículas absorvem a maior parte da luz incidente e o resto é refletido. Por fim, os raios de luz lançados atravessam o volume em uma única direção.

Max [1] propõe um modelo ótico partindo do princípio que os dados volumétricos consistem em um conjunto de pequenas partículas dentro de um

[1] *albedo* se refere à quantidade de luz refletida por uma partícula [16]

meio. Essas partículas, por sua vez, são as responsáveis pela emissão, absorção e reflexão da luz, e devem ser integradas continuamente ao longo de cada raio.

Serão vistos a seguir, os modelos de absorção, emissão e absorção mais emissão, baseando-se em Max [1]. As propriedades de reflexão e refração não foram tratadas neste trabalho e não foi adicionada nenhuma fonte externa de luz.

Para simplificar o cálculo das propriedades óticas a seguir, parte-se do princípio que todo volume sendo avaliado contém partículas esféricas idênticas de raio r e área $A = \pi r^2$. Além disso, considera-se que o número de partículas por unidade de volume é dado por ρ.

Absorção

Para calcular as propriedades de absorção, parte-se de um meio que contém apenas partículas escuras que absorvem toda luz incidente. Dessa forma, considerando um meio cilíndrico com base B, área E e altura Δs, parte-se do princípio onde a luz passa a ser transportada na direção perpendicular à base do cilindro, como ilustrado na Figura 2.5. Dessa forma, pode-se determinar que tal objeto cilíndrico possui um volume $E\Delta s$ e contém N partículas, sendo $N = \rho E \Delta s$.

Se for considerado que a altura da fatia Δs é pequena o suficiente para que não haja sobreposição na projeção das partículas contidas no objeto, pode-se calcular a área total de oclusão gerada por $NA = \rho A E \Delta s$. Dessa forma, a redução da intensidade de luz ao longo de um objeto cilíndrico B pode ser determinada por:

$$\Delta I = -\frac{NA}{E} I = -\frac{(\rho(s) E \Delta s) A}{E} I = -\rho(s) A I \Delta s \tag{2.5}$$

sendo I a luz emitida que está atravessando o volume. A intensidade da luz resultante após passar pelo objeto cilíndrico é dada por $I - \Delta I$. Fazendo a espessura da fatia se aproximar de zero, é possível determinar a equação diferencial através da quantidade de luz ocludida, tendo como princípio que nenhuma partícula irá se sobrepor a outra. A equação diferencial é dada por:

$$\frac{dI}{ds} = -\rho(s) A I(s) = -\tau(s) I(s) \tag{2.6}$$

tendo s como o parâmetro de distância ao longo do raio na direção do fluxo da luz, e $I(s)$ como a intensidade de luz na posição s. A solução da equação diferencial é dada por:

$$I(s) = I_0 e^{-\int_0^s \tau(t)\mathrm{d}t} = I_0 T(s) \tag{2.7}$$

onde $\tau(s) = \rho(s) A$ é o coeficiente de extinção, que define a taxa de luz obstruída em um ponto s (normalmente mapeado por uma função de transferência), e

I_0 corresponde à intensidade de luz na posição $s = 0$, onde o raio inicialmente intercepta o volume. Por fim, $T(s)$ é a transparência do meio entre 0 e s.

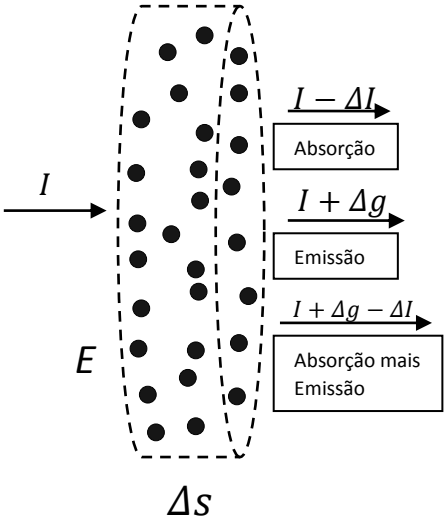


Figura 2.5 – Fatia de um meio em formato cilíndrico com base B de área E, altura Δs e ρ partículas por unidade de volume. Imagem adaptada de Max [1].

Em alguns casos o coeficiente de extinção é apenas determinado pela opacidade. Porém, sabendo que a transparência é dada por $T(s)$, a opacidade pode ser determinada por $\alpha = 1 - T(s)$. Dessa forma, quanto menos transparente for o meio, mais opaca será a imagem. A opacidade também é aproximada por $\alpha \approx \tau(s)d$, sendo d o intervalo que está sendo avaliado. Pode-se determinar $\tau(s)$ e α através de uma função de transferência (Seção 2.7).

Emissão

Assim como as partículas absorvem luz, elas também a emitem. Para o cálculo de emissão, as características de absorção das partículas são desconsideradas. Dessa forma, parte-se do princípio que as partículas da Figura 2.5 são transparentes e brilham com uma intensidade C por unidade de área projetada. A área projetada, determinada por $\rho AE\Delta s$, contribui com um fluxo de brilho $C\rho AE\Delta s$ para a área E, adicionando um fluxo de $C\rho A\Delta s$ por unidade de área. A equação diferencial resultante para calcular a intensidade de luz em um ponto s, $I(s)$, é dada por:

$$\frac{dI}{ds} = C(s)\rho(s)A_e = C(s)\tau(s) = g(s) \tag{2.8}$$

sendo $g(s)$ chamado de termo fonte, podendo incluir tanto características de emissão quanto de reflexão. A solução da equação diferencial (2.8) resulta na equação (2.9).

$$I(s) = I_0 + \int_0^s g(t)\mathrm{d}t \tag{2.9}$$

Absorção mais emissão

Este modelo leva em conta ambos os fatores de absorção e emissão. Neste caso, são consideradas partículas que não são nem totalmente escuras (com total absorção e sem emissão) nem totalmente transparentes (com total emissão e nenhuma absorção), possuindo uma certa opacidade e brilho. Dessa forma, as partículas acabam produzindo efeitos tanto de emissão quanto de absorção de luz. A equação diferencial do presente modelo leva em conta ambas as propriedades apresentadas anteriormente.

$$\frac{dI}{ds} = g(s) - \tau(s)I(s) \tag{2.10}$$

Dessa forma, definida por Williams e Max [17], a solução da equação (2.10) é descrita pela equação (2.11).

$$I(D) = I_0 T(D) + \int_0^D g(s)T'(s)\mathrm{d}s \tag{2.11}$$

sendo $T'(s) = e^{-\int_s^D \tau(x)\mathrm{d}x}$ e $T(s) = e^{-\int_0^s \tau(t)\mathrm{d}t}$. Na forma expandida, a equação (2.11) pode ser definida por:

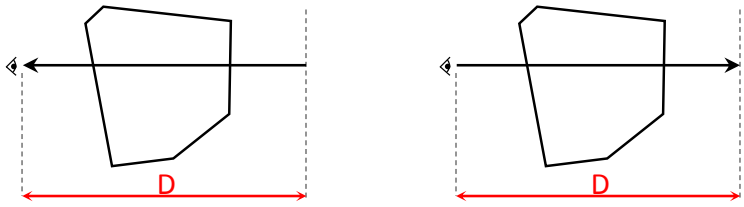
$$I(t_b) = I(t_f)e^{-\int_{t_f}^{t_b} \tau(f(t))\mathrm{d}t} + \int_{t_f}^{t_b} \tau(f(t))\kappa(f(t))e^{-\int_t^{t_b} \tau(f(u))\mathrm{d}u}\,\mathrm{d}t \tag{2.12}$$

onde t_f e t_b consistem no comprimento do raio do ponto do observador até, respectivamente, a face frontal e traseira interceptada; $f(t)$ é a função escalar reconstruída ao longo do raio que atravessa o volume; $\tau(f(t))$ é o fator de atenuação da luz, e $\kappa(f(t))$ é a intensidade luminosa. Geralmente, $\tau(f(t))$ e $\kappa(f(t))$ são determinadas por uma função de transferência: $\tau(f(t))$ se refere ao valor de opacidade e $\kappa(f(t))$ à um valor de cor. A integral volumétrica é, geralmente, avaliada para os canais R, G, B e A. Para o cálculo do canal A, o valor de $\kappa(s)$ é ignorado ($\kappa(s) = 1$).

A abordagem mais comum para aproximar a integral volumétrica é utilizar a soma de Riemann ao longo de n segmentos de mesmo tamanho $\Delta_x = L/n$ [18] (onde L é o tamanho do intervalo que será avaliado).

Engel et al. [19] descreveram a integral volumétrica de uma forma mais prática, assumindo o cenário em que, ao invés dos raios de luz chegarem até o observador (Figura 2.6a), raios são lançados a partir do ponto do observador (Figura 2.6b), acumulando as propriedades óticas do modelo. Dessa forma, a integral volumétrica pode ser descrita pela equação (2.13).

$$I = \int_0^D C(s(t))\tau(s(t))e^{-(\int_0^t \tau(s(t'))\mathrm{d}t')}\,\mathrm{d}t \tag{2.13}$$



(a) *back-to-front* (b) *front-to-back*

Figura 2.6 – Método de avaliação para as duas integrais apresentadas (equações (2.12) e (2.13)). Os raios de luz percorrem a distância D do final do volume até o ponto do observador em Figura 2.6a e os mesmos são lançados a partir do observador em Figura 2.6b.

onde D é a máxima distância de avaliação capturada pelo raio, $C(s(t))$ é a cor resultante dado um valor escalar $s(t)$ e $\tau(s(t))$ representa o coeficiente de extinção para um dado valor escalar $s(t)$, integrado ao longo do raio.

Utilizando a equação (2.13), o processo de implementação e entendimento do método acaba sendo mais intuitivo; principalmente quando se utiliza um algoritmo como o lançamento de raios para realizar a visualização volumétrica, como é o caso deste trabalho.

2.5 Discretização da integral volumétrica

Exceto em casos muito simples, não se consegue avaliar a integral volumétrica analiticamente [15]. Dessa forma, é realizada uma aproximação utilizando métodos de integração numérica, discretizando todo o tamanho de avaliação em subintervalos.

A discretização mais comum da integral volumétrica (2.13) apresentada na literatura é feita utilizando a soma de Riemann (2.14). Uma explicação detalhada dessa discretização pode ser vista no trabalho de Etiene et al. [20].

$$I = \sum_{i=0}^{n-1} C_i \tau_i d \prod_{j=0}^{i-1} (1 - \tau_j d) + O(d) \tag{2.14}$$

sendo i e j variáveis determinando a posição de uma amostra no raio; C_i é a cor emitida pela amostra i; τ_i representa o coeficiente de extinção; d é o tamanho de um passo e n o número total de passos dados. Dessa forma, sendo D o tamanho total da avaliação de um raio, podemos definir d por $d = D/n$. Pode-se abstrair a discretização da integral volumétrica para dois métodos numéricos quaisquer [21], obtendo:

$$\widetilde{T}(\lambda) = \prod_{j=0}^{n-1} e^{-\sum_{k=j}^{a} w_k(d)\tau_k} \tag{2.15}$$

$$T(\lambda) = \widetilde{T}(\lambda) + O(d^p) \tag{2.16}$$

$$I = \sum_{i=0}^{n-1} \sum_{k=i}^{b} \phi_k(d) C_k \tau_k T(k) + O(d^q) \tag{2.17}$$

onde b e a representam a quantidade de amostras necessárias para um dado método numérico aproximar um intervalo (3 para Simpson, 2 para trapézio, etc); $w_p(d)$ e $\phi_p(d)$ representam funções lineares de d; p e q correspondem à taxa de convergência (também chamado de *order of accuracy*) dos métodos de integração numérica utilizados para aproximar, respectivamente, a integral interna e externa.

Com a equação 2.17 pode-se calcular a integral volumétrica dado um método numérico para calcular a parte da integral interna e da integral externa. A avaliação da integral volumétrica pode ser feita a partir do cálculo da integral de cada forma geométrica que compõe o volume em questão (geralmente formados por tetraedros, hexaedros) ou através da captura de amostras ao longo de cada raio lançado.

2.6 Algoritmo de lançamento de raios

O algoritmo de lançamento de raios é um dos métodos utilizados em visualização volumétrica que consiste em lançar um ou mais raios para cada um dos pixels do plano de projeção, geralmente correspondendo ao tamanho tela. Dado cada raio lançado que intercepta o volume, este atravessa o mesmo realizando a captura de amostras ao longo da travessia.

De acordo com a bibliografia, a abordagem de lançamento de raios para visualização volumétrica foi inicialmente proposta por Blinn [16] e posteriormente foi melhorada por Garrity [14]. No trabalho de Blinn, cada raio lançado deve computar a absorção de luz por onde atravessa. No trabalho de Garrity, é realizada uma extensão do método proposto por Blinn, propondo um método para visualização de malhas não-estruturadas.

Pode-se perceber que, no algoritmo de lançamento de raios, uma das grandes características é a avaliação independente do raio. Dessa forma, o tempo utilizado para avaliar um raio não interfere no tempo utilizado para avaliar um outro raio lançado na cena. Além disso, o modo em que cada raio será avaliado e a velocidade do passo de avaliação de cada raio também são independentes. O pseudo-código do algoritmo de lançamento de raios é apresentado pelo Algoritmo 2.

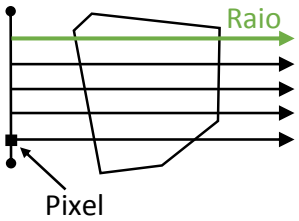


Figura 2.7 – Ilustração do algoritmo de lançamento de raios.

Algorithm 2 Visualização volumétrica baseada na estratégia de lançamento de raios.

```
1: for cada raio R da cena do
2:     I = 0
3:     for número de passos de tamanho D ao longo do raio R do
4:         I = IntegrarPasso(D, R, I)
5:     end for
6:     CalculaResultado(R, I)
7: end for
8: return I
```

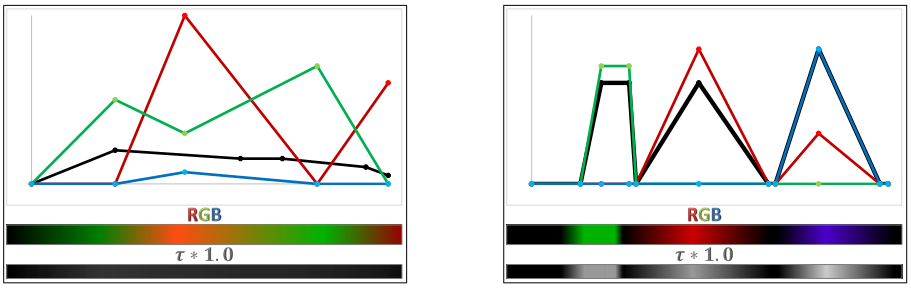


Figura 2.8 – Exemplos de função de transferência. Nos gráficos, foram determinados os três canais de cor RGB e o coeficiente de extinção.

2.7 Função de transferência

Uma função de transferência é responsável por atribuir propriedades visuais, como cor e opacidade, para os valores originais do campo escalar/vetorial de um dado volumétrico sendo visualizado [22]. Geralmente, ela é gerada como uma função linear por partes [23] gerada por pontos de controle, cada um contendo atributos de cor. A Figura 2.8 apresenta dois exemplos de função de transferência.

Um dos tópicos muito discutidos em visualização volumétrica é a geração automática de funções de transferência, tentando automatizar uma tarefa que é feita de maneira manual, na maioria das vezes. Algumas técnicas propostas utilizam histogramas LH [24] e geração de árvores de contorno [25]. A utilização de uma boa função de transferência é essencial para a boa interpretação de um modelo.

3
Trabalhos relacionados

Alguns tópicos de interesse foram trabalhados em visualização volumétrica ao longo dos anos. Dentre esses tópicos, serão vistos neste capítulo: métodos de integração numérica para avaliação da integral volumétrica (3.1); técnicas de visualização adaptativa (3.2); tabelas de pré-integração (3.3) e análise dos métodos de avaliação e discretização (3.4).

3.1
Métodos de integração numérica para avaliação da integral volumétrica

Como mencionado na Seção 2.5, não é possível avaliar a integral volumétrica analiticamente. Dessa forma, é necessário utilizar algum método de integração numérica para aproximar o resultado.

A maneira mais comum e tradicional de realizar a discretização da integral volumétrica é através da soma de Riemann [20]. Esse método é bastante utilizado pela simplicidade e, consequentemente, pelo alto desempenho. Porém, para se ter uma alta qualidade, muitas vezes é necessário diminuir muito o passo de avaliação. Visando melhorar a qualidade da avaliação, outros métodos de integração numérica de ordem maior foram também analisados.

Alguns dos métodos de integração numérica baseiam-se nas regras de quadratura Newton–Cotes, considerando a Regra do Trapézio [2] e Regra de Simpson [2, 26]. Novins e Arvo [2] buscaram um foco maior em avaliar intervalos de tamanhos adaptativos, enquanto Hajjar et al. [26] buscaram avaliar a integral volumétrica dividindo a travessia do raio em subintervalos iguais e aproximando-os utilizando a Regra de Horner para calcular polinômios de segundo grau.

Novins e Arvo [2] também apresentaram uma avaliação utilizando a série de potências (Figura 3.1). Miranda e Celes [5] exploraram a quadratura de Gauss–Legendre para a visualização de malhas não estruturadas, possibilitando a avaliação eficiente de polinômios com grau menor ou igual a $2n-1$, sendo n a quantidade de pontos (amostras) utilizados para realizar a aproximação.

3.2
Técnicas de visualização adaptativa

Um dos grandes desafios da visualização volumétrica é a utilização de técnicas eficientes para melhorar a avaliação da integral volumétrica. Em muitos casos, a avaliação é feita através da captura de amostras equidistantes

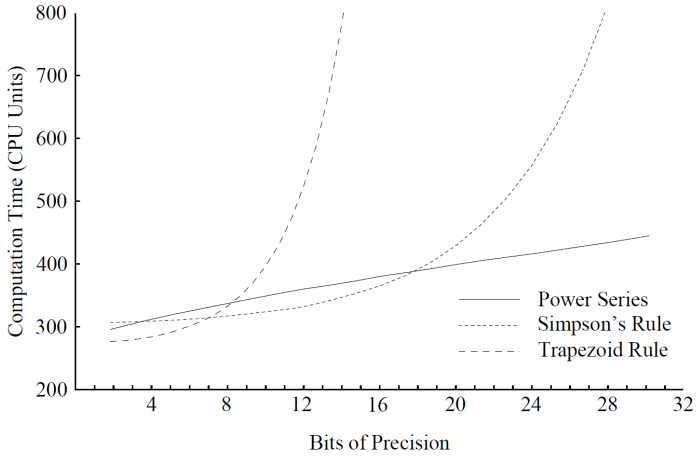


Figura 3.1 – Análise dos 3 métodos utilizados por Novins e Arvo [2] para calcular a integral volumétrica, considerando precisão por tempo de processamento.

ao longo de cada raio lançado, não levando em conta a variação do campo escalar. Dessa forma, a visualização pode acabar não capturando muitos detalhes do dado volumétrico em regiões de alta variação. Além disso, a captura de amostras adicionais em regiões constantes ou de pouca variação acaba sendo um desperdício de tempo, o que muitas vezes acontece quando são capturadas amostras equidistantes.

Técnicas de visualização adaptativa buscam direcionar o esforço necessário para avaliar a integral volumétrica, uma vez que o algoritmo acaba tratando cada uma das regiões de acordo com a sua importância/variação. Dessa forma, o esforço para avaliar um intervalo é menor em regiões onde o campo escalar possui baixa variação ou não gera uma contribuição significativa para o resultado final.

Considerando um cenário em que raios são lançados a partir do observador, pode-se considerar que cada um deles irá capturar o campo escalar do volume ao longo de sua travessia. Dessa forma, é possível determinar através do cálculo de derivadas qual a variação da função capturada pelo raio e, consequentemente, variar o tamanho do passo de avaliação.

Para que as derivadas sejam calculadas em um ponto arbitrário ao longo do raio, Novins e Arvo [2] propuseram pré-classificar o campo escalar do volume, visto que podem ocorrer descontinuidades nas fronteiras entre voxels (ou células). Um dos pontos negativos dessa abordagem é a quantidade de detalhe perdido por causa deste pré-processamento, além do próprio tempo necessário para gerar essa classificação. A função utilizada para pré-classificar o campo escalar depende do método de integração numérica utilizado.

Dada a funcionalidade de avaliar a n–ésima derivada em um intervalo, é possível determinar o tamanho de cada passo dado durante a avaliação da

integral volumétrica dada a variação do campo escalar. Novins e Arvo [2] procuraram utilizar essa propriedade para que, dado um valor de erro, seja possível determinar qual o valor do passo ideal. Da mesma forma, dado o tamanho do passo, é possível calcular qual o potencial erro que um certo intervalo de avaliado irá cometer (utilizando o termo remanescente dos métodos de integração abordados).

Novins e Arvo [2] propuseram uma estratégia de avaliação adaptativa utilizando o termo remanescente para controlar a avaliação da integral volumétrica para 3 tipos de integração numérica: Regra do Trapézio, Regra de Simpson e Série de Potências. Mesmo tendo um certo controle de erro durante a avaliação, utilizar apenas a avaliação das derivadas em um certo ponto para determinar o erro global do intervalo pode não ser a melhor das alternativas, uma vez que se verifica a variação do intervalo apenas em um ponto. Além disso, Novins et al. [27] relatam que, embora em [2] seja possível gerar uma imagem com alta precisão, a técnica não possui um bom equilíbrio entre tempo e precisão. Caso fosse realizada a avaliação de cada hexaedro de um volume, Hajjar et al. [26] citaram a possibilidade de adicionar uma amostra entre as extremidades de cada hexaedro e uma no ponto médio da travessia de cada um. Dessa forma, os dois intervalos poderiam ser interpretados como funções de segundo grau.

Um dos tópicos utilizados para adaptar o passo de avaliação é a construção de estruturas auxiliares que buscam aumentar o desempenho e qualidade da visualização. Estas estruturas tem por objetivo conseguir determinar regiões que devem demandar maior e menor esforço para serem avaliadas.

Em métodos que avaliam a integral volumétrica de maneira analítica (hexaedro por hexaedro), é comum calcular os pontos de máximo e mínimo (Figura 3.5), para que seja feita uma aproximação mais adequada. Com isso, é possível determinar algumas regras para que um pouco deste esforço seja poupado. Duas estratégias apresentadas são pré-determinar regiões invariáveis e pré-determinar visibilidade de células [28]:

1. **Pré-determinar regiões invariáveis:** Se o valor dos pontos extremos locais (mínimo e máximo) da função reconstruída pelo raio retornarem o mesmo valor, então não é necessário inserir quatro amostras, apenas uma em cada extremidade do intervalo. Isso é feito porque supõem-se que quando os pontos extremos forem iguais, a região mantém o mesmo valor durante todo o intervalo;

2. **Pré-determinar visibilidade de células:** Essa estratégia consiste em determinar uma tabela de visibilidade $VT(i, j)$ que define quando há

algum valor de opacidade diferente de zero entre os índices i e j mapeados na função de transferência. Dessa forma, é possível determinar quais células necessitam ser avaliadas e quais podem ser cortadas no processo de síntese por representarem uma região com opacidade nula (i e j normalmente correspondem aos pontos de máximo e mínimo).

Dessa forma, realiza-se a avaliação mais precisa apenas nas células que há necessidade. Porém, sempre haverá o gasto adicional de memória para que seja guardada essa tabela auxiliar. Wang et al. [28] realizam a quantização dos valores das amostras (geralmente em inteiros de 0 a 255) para que seja possível mapear a função de transferência e a tabela de visibilidade.

As estruturas baseadas em árvores buscam aumentar o desempenho da visualização, dividindo o modelo de acordo com suas respectivas regiões. Dessa forma, pode-se verificar quais regiões necessitam de um esforço maior para serem avaliadas. Marchesin e de Verdiere [3] propõem um método utilizando estruturas AMR (Figura 3.2). Utilizada para aumentar o desempenho da visualização, a estrutura AMR consegue isolar as regiões invariáveis, diminuindo a quantidade de amostras necessárias para realizar a avaliação do volume. Novins et al. [27] também propõem uma técnica que utiliza a árvore BSP (*Binary Space Partitioning*) alinhada aos eixos para determinar os intervalos de integração dos volumes. No caso desse trabalho, a árvore é refinada de acordo com a avaliação do erro de cada intervalo (determinado por cada folha da árvore) através do termo remanescente [2].

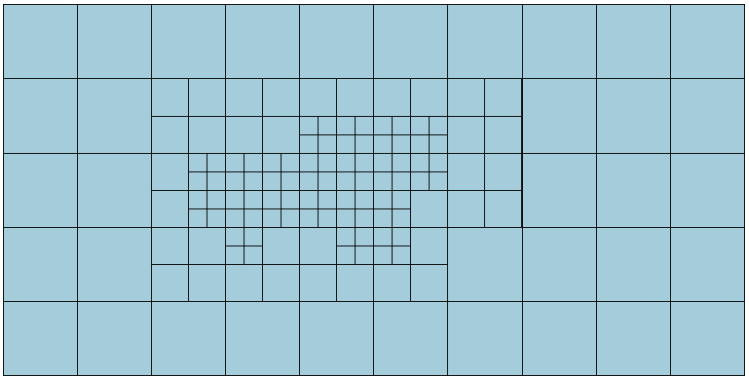


Figura 3.2 – Exemplo de estrutura AMR para um plano 2D (3 níveis de refinamento). Retirada de [3].

Outro modo de realizar a amostragem adaptativos em dados volumétrico baseia-se na geração geometria dos volumes. Lindholm et al. [4] propõem um método de amostragem adaptativa através da geração das isosuperfícies dos modelos volumétricos. Uma das principais vantagens deste método é a capacidade de não capturar amostras em regiões com opacidade nula.

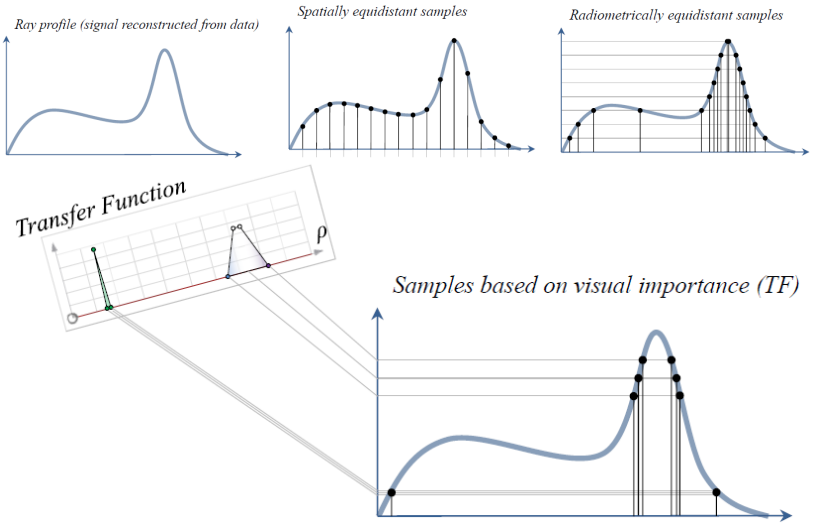


Figura 3.3 – Amostragem adaptativa baseada na variação do campo escalar e da função de transferência. Retirada de [4].

Porém, é necessário utilizar o algoritmo de *marching cubes* para determinar as isosuperfícies.

A ideia de Lindholm et al. é determinar um valor de distância variável entre as amostras na medida em que se aproxima de uma iso-superfície (Figura 3.3), diminuindo a quantidade de artefatos. Cada passo de avaliação é determinado através da distância entre duas isosuperfícies geradas. A principal limitação desse trabalho é ser necessário reconstruir toda a geometria caso haja alguma modificação na função de transferência. Além disso, é necessário utilizar um bloco adicional de memória para guardar a geometria das isosuperfícies geradas pelo algoritmo de *marching cubes* [29].

As funções de transferência são geralmente construídas a partir de pontos de controle, sendo classificadas como funções lineares por partes (Figuras 3.4 e 2.8). Cada um destes pontos de controle representa uma isosuperfície do modelo. Dessa forma, conseguir capturar amostras nas ocorrências dessas isosuperfícies resulta em uma avaliação mais precisa.

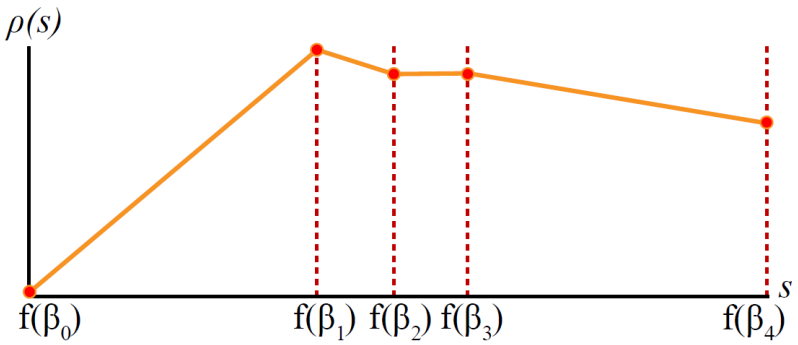


Figura 3.4 – Exemplo de função de transferência e seus pontos de controle. Retirada de [5].

Dada a Figura 3.4, supondo que um intervalo tem como valor inicial β_0 e valor final β_4. Se houver alguma maneira de identificar pontos de controle intermediários entre esses dois valores (β_1, β_2, β_3), pode-se calcular o intervalo de maneira mais precisa, através da subdivisão do mesmo em subintervalos baseados na função de transferência.

Williams et al. [23] utilizaram essa estratégia através do recorte de células para cada isosuperfície, tendo um foco maior em tetraedros com campo escalar linear. Moreland e Angel [30] estenderam a técnica utilizando processamento gráfico para determinar as isosuperfícies de tetraedros. Porém, assim como Moreland e Angel apontam, resulta em um custo adicional de memória, uma maior quantidade de instruções por fragmento e a necessidade de renderizar múltiplas vezes cada célula.

Miranda e Celes [5] implementaram esta estratégia para visualização de hexaedros. Nesse caso, a função escalar capturada por cada raio lançado é dividida em intervalos monotônicos, através da captura dos pontos de máximo e mínimo da função cúbica do hexaedro [5, 28] (Figura 3.5). Dessa forma, é possível verificar em cada intervalo monotônico as ocorrências de isosuperfícies.

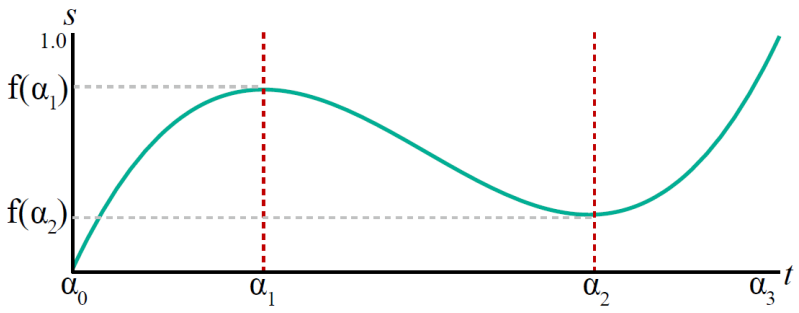


Figura 3.5 – Função capturada por um raio que atravesso o campo escalar de um hexaedro. α_1 e α_2 são respectivamente os pontos de máximo e mínimo do intervalo. Figura retirada de [5].

Essa estratégia aumenta a qualidade da visualização, porém acaba gerando um grande consumo de tempo. Ela é comumente utilizada com tabelas de pré-integração, que são geradas em pré-processamento (Seção 3.3).

Geralmente, técnicas de visualização adaptativa conseguem contribuir para aumentar o desempenho do algoritmo, encontrando as regiões que precisam de um maior esforço para serem avaliadas corretamente. De fato, a maioria dos dados volumétricos possuem regiões com uma variação mais baixa e regiões com variação mais alta. Um dos grandes desafios encontrados é conseguir adaptar o passo de avaliação de forma eficiente e determinar uma boa heurística de adaptatividade do passo. Um dos pontos não muito discutidos na literatura de visualização volumétrica se refere às funções aplicadas para determinar a simi-

laridade entre duas amostras e, consequentemente, o erro entre duas avaliações de um determinado intervalo.

3.3 Tabelas de pré-integração

Em visualização volumétrica, o tempo para processar um quadro da visualização é um dos grandes desafios, principalmente quando se necessita de uma maior precisão. Essa precisão vem da avaliação mais detalhada dos intervalos que serão capturados ao longo de cada raio lançado.

Para isso, foi proposta uma técnica de pré-processamento que consiste em pré-calcular todos os conjuntos de intervalos possíveis e guardá-los numa estrutura auxiliar indexada [31, 32]. Quando for necessário saber qual o valor da integral aproximada de um certo intervalo durante a visualização, basta procurar nesta estrutura gerada em pré-processamento. Esta estratégia é comumente chamada de *Pre-Integrated Volume Rendering* [19, 26, 33], gerando uma tabela com dimensão variável.

A tabela consiste em intervalos de avaliação calculados antes de renderizar o volume. Dessa forma, para realizar a integração do volume basta utilizar estes intervalos pré-calculados e para calcular a avaliação da integral volumétrica (2.13) para cada pixel da tela.

Normalmente, o acesso aos valores de uma tabela é dado pelo valor inicial, final e pelo tamanho do intervalo, resultando em uma estrutura 3D. Se fossem utilizados polinômios de segundo grau seria necessário uma tabela 4D [26], pela necessidade de adicionar uma amostra no ponto médio do intervalo. Caso as amostras sejam equidistantes, o campo de distância pode ser ignorado, retornando para uma tabela 3D [26].

Uma das grandes vantagens da tabela de pré-integração é precisar calcular os intervalos apenas uma vez. Além disso, pode-se calcular esses intervalos com alta precisão, visto que a tabela normalmente é gerada em pré-processamento. Por outro lado, haverá sempre um custo adicional de memória.

Normalmente, é necessário recalcular toda a tabela caso ocorra alguma modificação na função de transferência. Moreland e Angel [30] propuseram uma estrutura que calcula parcialmente os intervalos da integral volumétrica, chamada de *partial pre-integration*. Dessa forma, não é mais necessário recalcular toda a tabela dada uma modificação na função de transferência, partindo que a mesma é construída a partir de funções lineares por partes [23].

Visto que o objetivo do presente trabalho é gerar uma visualização de maneira adaptativa, utilizar as tabelas de pré-integração é um problema mais complexo, pois necessita-se adicionar um parâmetro na tabela para guardar

o tamanho do passo. Além disso, é necessária alguma estratégia para cobrir todos os tamanhos de intervalos possíveis.

3.4 Análise dos métodos de avaliação e discretização

Dentre vários aspectos a serem considerados na corretude dos algoritmos de visualização volumétrica, um dos mais importantes é a aproximação da integral volumétrica [20]. A avaliação incorreta da mesma pode gerar uma grande quantidade de erros e, consequentemente, acabam gerando erros visuais; resultados menos precisos e, por fim, uma percepção mais pobre do conjunto de dados.

Alguns trabalhos buscaram avaliar a quantidade de erro gerada e a eficiência de algumas aproximações numéricas [20, 26, 34]. De fato, é importante verificar quais são as consequências de utilizar um método numérico para gerar um resultado em tempo viável. Existem 4 técnicas principais utilizadas para a avaliação desses métodos [20, 34]:

- Julgamento de especialista (*Expert judgment*): Um especialista verifica se o resultado gerado de uma implementação está correto ou não;
- Quantificação do erro (*Error quantification*): O erro é quantificado através da avaliação dos resultados gerados com o resultado real ou com algum tipo de *ground-truth*;
- Análise de convergência (*Convergence analysis*): Processo que verifica se o erro cometido pelas aproximações feitas converge para zero, dado um certo parâmetro sendo variado (tamanho do passo de avaliação, por exemplo);
- Ordem de precisão (*Order-of-accuracy*): Verificar se a velocidade em que os erros cometidos diminuem, está de acordo com o comportamento estimado.

Regras mais conhecidas como a do trapézio e de Simpson possibilitam uma aproximação mais precisa de polinômios, resultando em uma avaliação mais precisa e suave da integral volumétrica. Um dos tópicos discutidos em relação aos métodos de integração numérica é a análise da corretude alcançada pelos diferentes tipos de aproximação numérica. Uma das maneiras mais simples é quantificar o erro dada uma função conhecida [26]. Hajjar et al. [26] buscaram mostrar que a regra de Simpson é capaz de gerar resultados mais suaves com uma menor quantidade de erro, devido à utilização de um polinômio de maior grau para aproximar um intervalo de avaliação.

Etiene et al. [20] propuseram uma metodologia de avaliação da corretude dos métodos de discretização da integral volumétrica utilizando as abordagens de análise de convergência e ordem de precisão. A avaliação é feita através da modificação de parâmetros (distância de cada amostra, número de pixels, tamanho de cada célula) ao longo de uma avaliação, focando na avaliação com a soma de Riemann. Posteriormente, Etiene et al. [21] estenderam a análise para outros métodos de integração numérica conhecidos, avaliando a taxa de convergência de cada um deles.

De acordo com os autores em [21, 26], foi visto que a regra de Simpson gera resultados bem precisos, tendo um bom fator de convergência. Pelo fato do hexaedro gerar um campo de variação trilinear, a regra de Simpson parece ser uma boa estratégia para realizar uma aproximação adequada.

Quando se trabalha com malhas não-estruturadas, o campo escalar pode ser simplificado através da subdivisão das células. Carr et al. [35] realizaram uma comparação dos métodos de subdivisão de malhas não estruturadas, mostrando as limitações de cada estratégia. As subdivisão são feitas de malhas de hexaedros para malhas de tetraedros. Visto que o campo escalar acaba de modificando devido às formas de interpolação utilizadas, é feita a análise das características de cada uma dessas subdivisões e o erro gerado a partir delas.

Carr et al. [35] relataram que a simplificação de malhas em tetraedros possui suas vantagens: podem ser geradas em tempo de execução; tem apenas 3 casos de iso-superfícies (contra 15 do método de *marching cubes*); os pontos críticos de cada tetraedro encontram-se nos vértices e o campo escalar é monotônico. Porém, essas vantagens tem um preço. Além de aumentar o tamanho da malha pela quantidade de tetraedros, geralmente não se consegue capturar de maneira eficaz o campo escalar de variação trilinear dos hexaedros, gerando artefatos visuais.

4
Algoritmo proposto

Este trabalho foi executado com um caráter exploratório e experimental. O algoritmo proposto baseia-se na técnica de lançamento de raios, com um raio sendo lançado por pixel. Essa técnica foi escolhida pelo fato de cada raio ser avaliado de maneira independente e por ser um método altamente paralelizável. Além disso, é o algoritmo que pareceu viabilizar melhor a implementação de passos adaptativos.

A abordagem adaptativa implementada neste trabalho baseia-se na conhecida regra de Simpson e no método de Simpson adaptativo. A equação da integral volumétrica é composta por duas integrais, determinadas neste trabalho como integral interna ($\int_{[I]}$) e externa ($\int_{[E]}$), sendo $\int_{[E]} = f(\int_{[I]})$. Visto que o método se propõe a manter o erro controlado, tanto a integral externa quanto a interna são avaliadas de maneira adaptativa, realizando recorrentes avaliações de erro até que o intervalo esteja dentro da tolerância definida.

A abordagem mais simples de implementar o método de Simpson adaptativo para visualização de um dado volumétrico é, dado o ponto inicial e final da travessia do raio, aplicar diretamente o algoritmo recursivo de Simpson adaptativo. Para cada intervalo, realiza-se a avaliação da integral externa $\int_{[E]}$; caso o erro estimado seja maior que a tolerância pré-definida, o intervalo é dividido em duas metades e é chamada a recursão para a avaliação dos subintervalos.

O principal problema dessa abordagem simples é a necessidade de também avaliar de maneira adaptativa a integral interna, pois toda avaliação da função externa em um ponto necessita da avaliação da integral interna do início da integração até o ponto em questão que está sendo avaliado. Essa abordagem acaba gerando uma grande quantidade de recálculos de intervalos porque não é possível guardar todos os resultados intermediários das avaliações de intervalos da integral interna. Além disso, o processo recursivo impossibilita o mapeamente eficiente do método em GPU.

Dessa forma, foi implementado um algoritmo alternativo, tornando o processo recursivo em um processo iterativo. Com isso, o método passa a controlar de maneira separada, porém coerente, o tamanho do passo para o cálculo de ambas as integrais. Este processo iterativo baseia-se nos métodos de resolução numérica de soluções de equações diferenciais ordinárias (método de Runge-Kutta, por exemplo). O processo consiste em trocar todas as chamadas de recursão por um laço global, cujo critério de parada é atingir o ponto final da travessia do raio.

O Algoritmo 3 ilustra o método desenvolvido neste trabalho. O pseudo-código descreve o procedimento principal realizado tanto para o cálculo da integral interna quanto para o cálculo da integral externa.

Algorithm 3 Método iterativo proposto.

Input: $t, \Delta t, h_0, \epsilon$
Output: $\int(t, \Delta t, h_0)$

$I = 0$
$t_{final} = t + \Delta t$
$h = \min(h_0, t_{final} - t)$
$lastsucceeded = true$
while $t < t_{final}$ **do**
  if $E(t,h) \leq \epsilon$ **then**
    $I = I + D(t,h) + E(t,h)$
    $t = t + h$
    if $lastsucceeded$ **then**
      $h = 2h$ {Incrementa o passo}
      $\epsilon = 2\epsilon$
    else
      $lastsucceeded = true$
    end if
    $h = \min(h, t_{final} - t)$
  else
    $h = h/2$ {Decrementa o passo}
    $\epsilon = \epsilon/2$
    $lastsucceeded = false$
  end if
end while
return I

Dado um intervalo Δt, o objetivo do método implementado é avaliar a integral volumétrica (2.13) de t até $t_{final} = t + \Delta t$. O método também recebe o tamanho do passo inicial, $h = h_0$, utilizado no início do procedimento para começar a evolução da integração.

Dessa forma, utiliza-se a regra de Simpson Adaptativa para avaliar a integral (2.1) e o respectivo erro estimado (2.4). Caso o erro seja maior do que a tolerância exigida, o tamanho do passo é diminuído pela sua metade e realiza-se uma nova tentativa de avaliação, porém agora avaliando o passo de integração de t até $t + \frac{h}{2}$. Caso contrário, se o resultado estiver dentro da tolerância (ou seja, se o erro estimado for menor que a tolerância exigida), o resultado é acumulado (2.2) e a integração é avançada para $t_{new} = t + h$. Na próxima iteração, avalia-se o próximo intervalo, de t_{new} até $t_{new} + h_{new}$. O novo passo adotado, h_{new} segue a seguinte regra: se houver pelo menos dois sucessos consecutivos de avaliação (sem a necessidade de subdividir o

intervalo), o tamanho do passo é dobrado; caso contrário, o tamanho do passo é mantido.

A Figura 4.1 ilustra como o passo é sendo adaptado: as linhas pontilhadas determinam as falhas, enquanto as linhas sólidas indicam as tentativas que obtiveram sucesso. O número de cada linha indica a ordem de cada tentativa. Neste caso ilustrativo, pode-se perceber que tanto o método recursivo quanto o método iterativo proposto irão processar as mesmas tentativas, resultando na mesma configuração final de integração (mesmos intervalos).

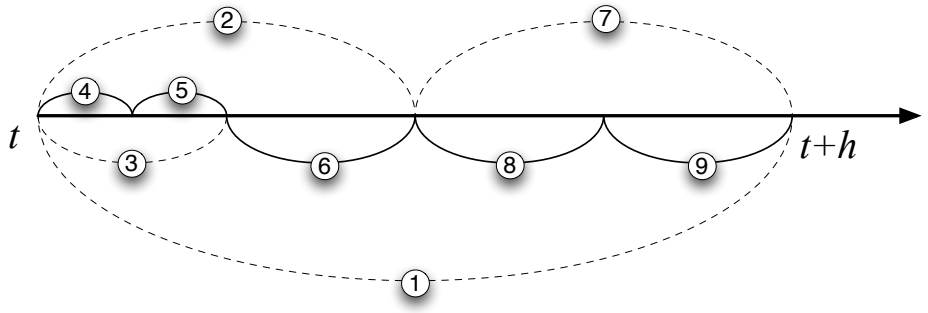


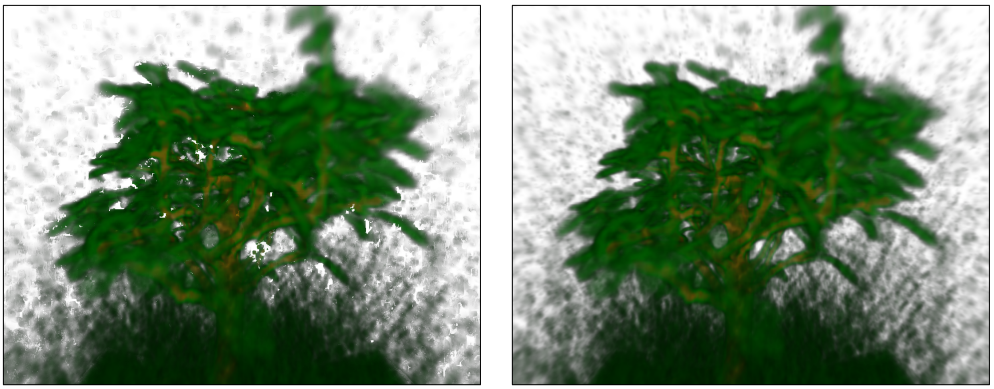
Figura 4.1 – Exemplo de como o método proposto controla o tamanho do passo ao longo da integração.

O Algoritmo 3, iterativo, foi utilizado de base para calcular a integral volumétrica. Na implementação foram utilizados dois parâmetros adicionais para controlar o intervalo de variação do passo, sendo eles h_{min} e h_{max}. Dessa forma, se o passo for menor que o valor mínimo h_{min}, o passo é considerado como dentro da tolerância (independente do valor de erro calculado). Caso contrário, se o passo for maior que o valor máximo tolerado h_{max}, o passo é reduzido ao valor de h_{max} (antes de realizar qualquer avaliação de erro, este teste é feito).

O passo mínimo foi utilizado como uma forma de prevenir uma redução do passo de maneira desnecessária, muitas vezes porque o erro não pode ser calculado de uma maneira precisa. No caso do passo máximo, ele é necessário para que não haja uma perda muito grande em regiões com alta variação em um espaço pequeno. A Figura 4.2 ilustra a importância de limitar o crescimento do passo (nota-se a diferença nas folhas do Bonsai).

Para calcular a integral volumétrica de maneira iterativa e adaptativa, primeiramente se avalia o tamanho do passo dado na integral interna, garantindo que este intervalo esteja dentro da tolerância permitida. Para calcular a integral interna, $\int_{[I]}$, utiliza-se o Algoritmo 3 com pequenas modificações. Dentre elas:

– Utilizam-se as variáveis h_{min} e h_{max};



(a) Sem h_{max} (b) Com $h_{max} = 8.0$

Figura 4.2 – Exemplo da importância de limitar o passo máximo. Constata-se que o Bonsai da Figura 4.2a contém erro de avaliação nas folhas, onde não se limita o passo máximo. A Figura 4.2b é gerada com $h_{max} = 8.0$.

- O processo termina assim que for encontrado um primeiro passo válido;
- É retornado o tamanho do passo e o valor aproximado da integração.

O valor $\int_{[I]}$ calculado é utilizado essencialmente para determinar se é necessária ou não a avaliação da integral externa $\int_{[E]}$, pois caso $\int_{[I]}$ seja igual a 0, pode-se constatar que o meio que está sendo percorrido é transparente e não irá afetar no resultado final. Nos experimentos realizados, a parte da integral externa é apenas ignorada caso o valor de $\int_{[I]}$ seja igual a 0.

O valor mais importante a ser considerado no processo de avaliação da integral interna é o tamanho do passo que foi dado. Este passo garante que, dentro deste intervalo, a integral interna pode ser avaliada de maneira precisa (dentro da tolerância exigida). Tendo este passo calculado, realiza-se o cálculo da integral externa $\int_{[E]}$ neste intervalo, utilizando o Algoritmo 3 com algumas pequenas modificações. Dentre elas:

- Também utilizam-se as variáveis h_{min} e h_{max};
- A integração é calculada para todos os canais de cor ($RGBA$);
- O algoritmo itera até avaliar todo o intervalo;
- Além de retornar o valor aproximado da integração, também é retornado o tamanho do último passo dado.

O principal problema da abordagem mais simples está na necessidade de recalcular a integral interna a cada avaliação da integral externa. Já o algoritmo proposto parte do princípio de que, dado um passo h com erro controlado, qualquer passo $h_n \leq h$ tem erro controlado. Dessa forma, dado um passo h

avaliado pelo método de simpson adaptativo com erro controlado, determina-se que toda regra de simpson aplicada ao longo deste intervalo também contém erro controlado.

Sendo assim, dentro do processo de avaliação da integral externa $\int_{[E]}$ em um ponto, é utilizada a regra de Simpson (não-adaptativa) para avaliar a integral interna quando necessário, visto que o passo da integral interna válido já foi calculado antes de realizar qualquer avaliação na integral externa.

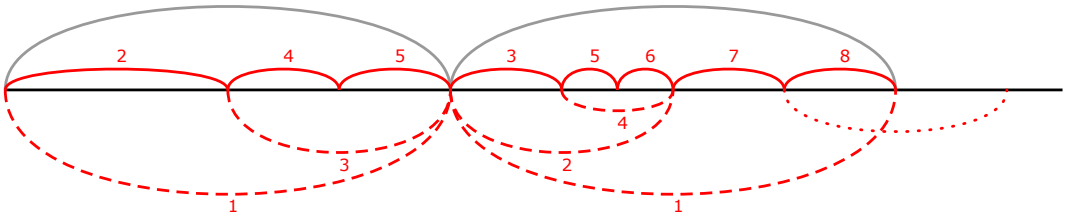
O pseudo-código do método de avaliação adaptativa é demonstrado pelo Algoritmo 4. Vale lembrar que, o valor de tolerância, omitido nas chamadas de função da integral interna e externa, é ajustado de acordo com o tamanho do passo em relação ao tamanho total do intervalo a ser avaliado.

Algorithm 4 Proposta de avaliação adaptativa da Integral Volumétrica.

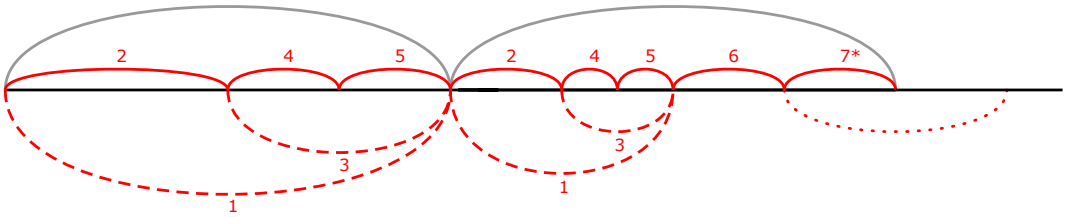
Input: $t, \Delta t, h_0, \epsilon$
Output: VRI $(t, \Delta t)$
1: $I = [R = 0, G = 0, B = 0, A = 0]$
2: $t_{final} = t + \Delta t$
3: $h_{int} = \min(h_0, t_{final} - t)$
4: $h_{ext} = h_{int}$ {Apenas na estratégia Desacoplada}
5: **while** $t < t_{final}$ **do**
6: $I_{int}, h_{int} = \int_{[I]}(t, t_{final} - t, h_{int})$
7: $h_{ext} = h_{int}$ {Apenas para a estratégia Acoplada}
8: **if** NOT $(h_{ext} \geq h_{int}$ AND $I_{int} = 0)$ **then**
9: $I_{ext}, h_{ext} = \int_{[E]}(t, h_{int}, h_{ext})$
10: $I = I + I_{ext}$
11: **end if**
12: **end while**
13: **return** I

Uma questão importante na avaliação da integral volumétrica é como controlar o passo utilizado para avaliar o passo da integral interna e externa. Foram experimentadas duas estratégias diferentes (como indicado no Algoritmo 4), nomeadas de Acoplada e Desacoplada.

Na estratégia Acoplada, o passo calculado em cada iteração da integral interna é utilizado de base para determinar o passo inicial da integral externa. Essa estratégia foi proposta baseando-se na hipótese de que as duas integrais variam da mesma forma. Dessa forma, tenta-se integrar todo o intervalo em uma tentativa inicial. Além disso, pode-se começar a avaliação da integral externa com o mesmo passo válido calculado na última avaliação da integral interna, lembrando também que este é o valor total do passo que deve ser integrado nessa iteração. O fato de utilizar o mesmo passo da integral interna também auxilia na reutilização de amostras, não sendo necessário acessar



(a) Estratégia Acoplada



(b) Estratégia Desacoplada

Figura 4.3 – Exemplo de sequência de passos da integral externa (em vermelho), dados dois passos de integração da integral interna (em cinza) de mesmo tamanho, para as duas estratégias de controle de passo apresentadas.

novamente o modelo e a função de transferência para recalcular as amostras nesta primeira avaliação.

Na estratégia Desacoplada, o passo da integral externa é controlado de maneira independente. Porém, o passo da integral interna ainda é utilizado para determinar um limite para o passo da integral externa, visto que o método se propõem a manter o erro controlado para as duas integrais. Essa estratégia procura apresentar uma coerência espacial mais inteligente do que a primeira estratégia. Dada uma função de transferência, caso a variação da opacidade seja baixa, porém a variação da cor seja alta, a estratégia Desacoplada acaba se comportando de maneira mais coerente.

A Figura 4.3 ilustra um pouco como o controle de passo da integral externa é feito para cada uma das estratégias a partir de dois passos de integração dados pela integral interna (em cinza). Para fins de melhorar o entendimento, partiu-se de um cenário inicial em que o primeiro intervalo obteve a mesma configuração, mudando o comportamento para cada estratégia no segundo passo. As linhas tracejadas são decorrentes dos passos de falharam enquanto as linhas sólidas representam os passos que obtiveram sucesso.

Como pode-se perceber, no segundo passo de integração, a estratégia Acoplada simplesmente determina que o passo da integral externa começa sendo o mesmo valor do passo da integral interna, independente do último passo que foi dado em um intervalo anterior. Já na estratégia Desacoplada, utilizou-se do histórico de passos dados no intervalo anterior para determinar qual seria o próximo passo para calcular o intervalo seguinte. No caso do exemplo da Figura 4.3b, os passos 4 e 5 do primeiro intervalo obtiveram sucesso, ou seja,

dois sucessos consecutivos. Então, tentou-se dobrar o passo de acordo com o algoritmo adaptativo implementado.

Ainda na Figura 4.3b, percebe-se que, no segundo intervalo, há um passo pontilhado e outro com a numeração 7 contendo um asterisco. Esse é o caso em que o passo que deveria ser dado é maior do que o tamanho restante do intervalo. Nesse caso, tenta-se integrar este passo remanescente sem mudar o valor original do passo da integral externa h_{ext}. Esse passo, por sua vez, só será atualizado caso haja a necessidade de diminuir o passo remanescente pela metade, caso em que o erro fica acima da tolerância pedida. Vale lembrar que, para a estratégia Desacoplada, o passo da integral externa também possui o seu respectivo valor de passo inicial $h_{[0,ext]}$. Nos resultados, foi utilizado o mesmo valor da integral interna.

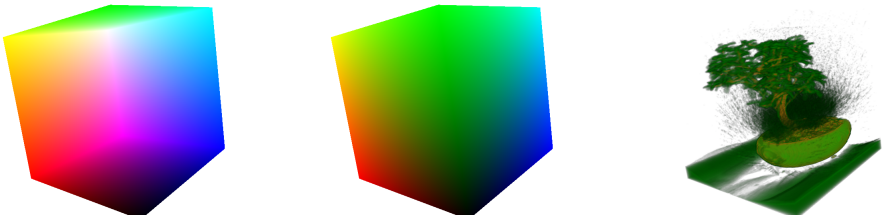
Nos resultados gerados, as duas estratégias acabaram sendo praticamente equivalentes. A estratégia Acoplada acabou sendo um pouco mais eficiente, enquanto a estratégia Desacoplada acabou sendo um pouco mais precisa. De fato, pela Figura 4.3 pôde-se perceber que a estratégia Desacoplada procura manter uma coerência espacial do passo da integral externa enquanto a estratégia Acoplada busca um ganho maior de desempenho, negligenciando as últimas ocorrências dos passos adaptativos para saber se a integração poderia estar em uma região de maior variação.

Percebe-se, no algoritmo, que um dos parâmetros passados para o método é o valor inicial do passo (h_0). Mudar o tamanho do passo inicial irá apenas interferir na performance da avaliação, ainda de forma pouco expressiva. Pelo fato do método ter controle do erro cometido, pode-se utilizar qualquer valor como passo inicial. No entanto, sabe que assim como a integração, a avaliação de erro é baseada em uma estratégia de captura de amostras. Então, regiões que apresentam alta variação ao longo de intervalo podem ser erroneamente ignoradas.

4.1 Implementação

O algoritmo foi implementado em C++ para CPU e GPU. Em GPU, também foi utilizada a linguagem GLSL. Foram utilizados modelos compostos por malhas estruturadas, devido à fácil leitura e manipulação dos dados. Além disso, pode ser representada por uma estrutura de dados simples (um array contendo as intensidades e os valores de comprimento, altura e profundidade do dado volumétrico).

Em CPU, verifica-se se cada raio lançado atravessou o modelo através de uma função de interseção. Em GPU, a implementação baseia-se em um



(a) *Back faces* (b) *Front faces* (c) Saída

Figura 4.4 – Principais passos do algoritmo de visualização volumétrica implementado em GPU.

algoritmo de duas passadas (Figura 4.4), utilizando um *framebuffer* para armazenar as informações das faces da caixa envolvente do dado volumétrico.

Na primeira passada são desenhadas apenas as *back faces* (Figura 4.4a) da caixa envolvente do modelo, armazenando-as em um *framebuffer*. Na segunda passada, são geradas as *front faces* da caixa envolvente (Figura 4.4b) e realiza-se a avaliação e visualização do volume no *fragment shader* (Figura 4.4c).

A geração das *front faces* (Figura 4.4b) e *back faces* (Figura 4.4a) da caixa envolvente do modelo geram um mapeamento das coordenadas em valores RGB. Dessa forma, pode-se calcular o ponto inicial e final da travessia de cada raio.

Foi utilizada a interpolação trilinear para calcular o valor de cada amostra em um ponto qualquer do volume. Em CPU esse procedimento toma um certo tempo de processamento, pois deve ser feito o cálculo para cada amostra. Já em GPU têm-se isso nativamente pela placa gráfica, o que torna o processo de avaliação de amostras mais rápido devido ao cache.

No algoritmo original (recursivo), as amostras adjacentes e as amostras no ponto médio são compartilhadas caso haja a necessidade de subdividir o intervalo para reavaliá-lo recursivamente, diminuindo o tempo gasto em capturar amostras. O preço da reutilização de amostras e avaliações de função acaba sendo pago pela quantidade de memória adicional utilizada com as chamadas recursivas. No caso do algoritmo iterativo, seria necessário implementar uma pilha para guardar os valores. Dessa forma, preferiu-se apenas reutilizar as amostras da primeira metade de um passo e as amostras da segunda metade devem ser calculadas novamente.

Mesmo tendo duas integrais diferentes para serem avaliadas, foi decidido utilizar um valor único de tolerância tanto para avaliar a integral interna quanto para avaliar a integral externa. Além disso, foram utilizadas funções de transferência lineares por partes, pela simplicidade de armazenamento e manipulação.

Para calcular o erro de avaliação da integral externa, foi utilizado o valor

absoluto da subtração de cada canal de cor RGBA, avaliando o erro de cada canal de maneira independente. Sendo assim, se um dos canais apresentar um erro maior que o tolerável, o algoritmo irá continuar a subdivisão dos intervalos.

5
Resultados e discussão

Foram realizados testes de desempenho e precisão do método adaptativo implementado. Os testes foram gerados em um computador Intel Core 2 Quad 2.66 GHz, 4 GB de RAM, com uma placa de vídeo GeForce GTX 560. Foram utilizados dados volumétricos de malhas estruturadas, obtidos através do site volvis.org [36] e NASA [37]. As informações gerais dos modelos utilizados estão presentes na Tabela 5.1. Os resultados foram gerados a partir das implementações em CPU, com precisão dupla (tipo *double*), e GPU, com precisão simples (tipo *float*), com uma tela de projeção de dimensão 800×600. Informações adicionais dos resultados e resultados adicionais gerados encontram-se nos apêndices B, C e D.

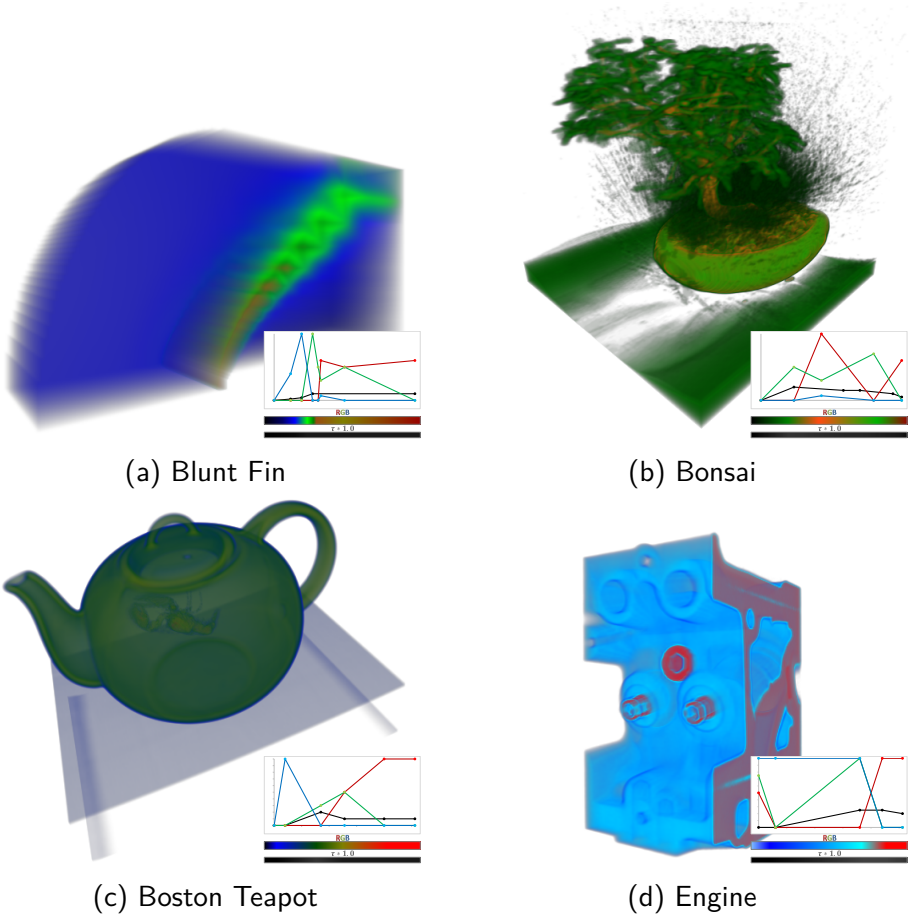
As funções de transferência utilizadas na geração dos resultados foram geradas a partir de pontos de controle, interpolando as regiões intermediárias. Elas mapeiam para cada valor escalar, um valor de cor RGB e um valor correspondente ao coeficiente de extinção (τ). Foi utilizado o coeficiente de extinção ao invés da opacidade em si porque não serão integrados apenas intervalos equidistantes.

Foi utilizado como método de comparação dos resultados, tanto de desempenho e precisão, a discretização da Integral Volumétrica utilizando a Soma de Riemann (2.14), variando o passo em 0.5, 0.4 ,0.3, 0.2 e 0.1 (em unidade de *voxel*). Para os teste de precisão, foi determinado o passo de 0.01 (em unidade de *voxel*) como resultado correto da visualização (valor de referência).

Os valores de ϵ, h_{min} e h_{max} afetam diretamente o resultado gerado pelo método adaptativo. Diferentes modelos obtém resultados melhores a partir da correta configuração destes parâmetros para cada um dos casos. Inicialmente, foram gerados resultados com uma configuração única (Figura 5.1) para fins de simplificar o experimento. A partir dos primeiros resultados, foi gerada uma segunda configuração para o modelo Bonsai (Figura 5.1b), buscando melhorar

Tabela 5.1 – Informações gerais dos modelos testados.

Nome	Dimensões	Raios avaliados (CPU)
Blunt Fin	256×128×64	268207
Bonsai	256×256×256	223092
Boston Teapot	256×256×178	270101
Engine	256×256×128	177373



(a) Blunt Fin (b) Bonsai

(c) Boston Teapot (d) Engine

Figura 5.1 – Visualização dos modelos utilizados. Para a geração das imagens foram utilizados: $h_0 = 0.5$, $h_{min} = 0.1$, $h_{max} = 2.0$ (em unidade de *voxel*) e $\epsilon = 0.01$. Utilizou-se a estratégia Acoplada para o controle do passo.

o desempenho e mantendo alta precisão.

Nos primeiros testes, foram utilizados: $h_0 = 0.5$, $h_{min} = 0.1$, $h_{max} = 2.0$ (em unidade de *voxel*), variando a tolerância ϵ entre 0.01, 0.005 e 0.001. Foram testadas as duas estratégias de controle de passo propostas: Acoplada (*Coupled*) e Desacoplada (*Decoupled*). Os parâmetros h_{min} e h_{max} foram ajustados dessa forma devido à necessidade de garantir precisão no modelo Bonsai (Figura 5.1b), diminuindo o valor de h_{min}, e devido à grande região constante com opacidade baixa do modelo Blunt Fin (Figura 5.1a), diminuindo o valor de h_{max}

Os testes de precisão foram feitos em CPU a partir da diferença absoluta de cada canal de cor (R, G, B e A), comparando com a referência escolhida. Dessa forma, verifica-se, para cada raio, se o erro gerado é maior do que o valor de tolerância escolhida (0.01, 0.005 ou 0.001).

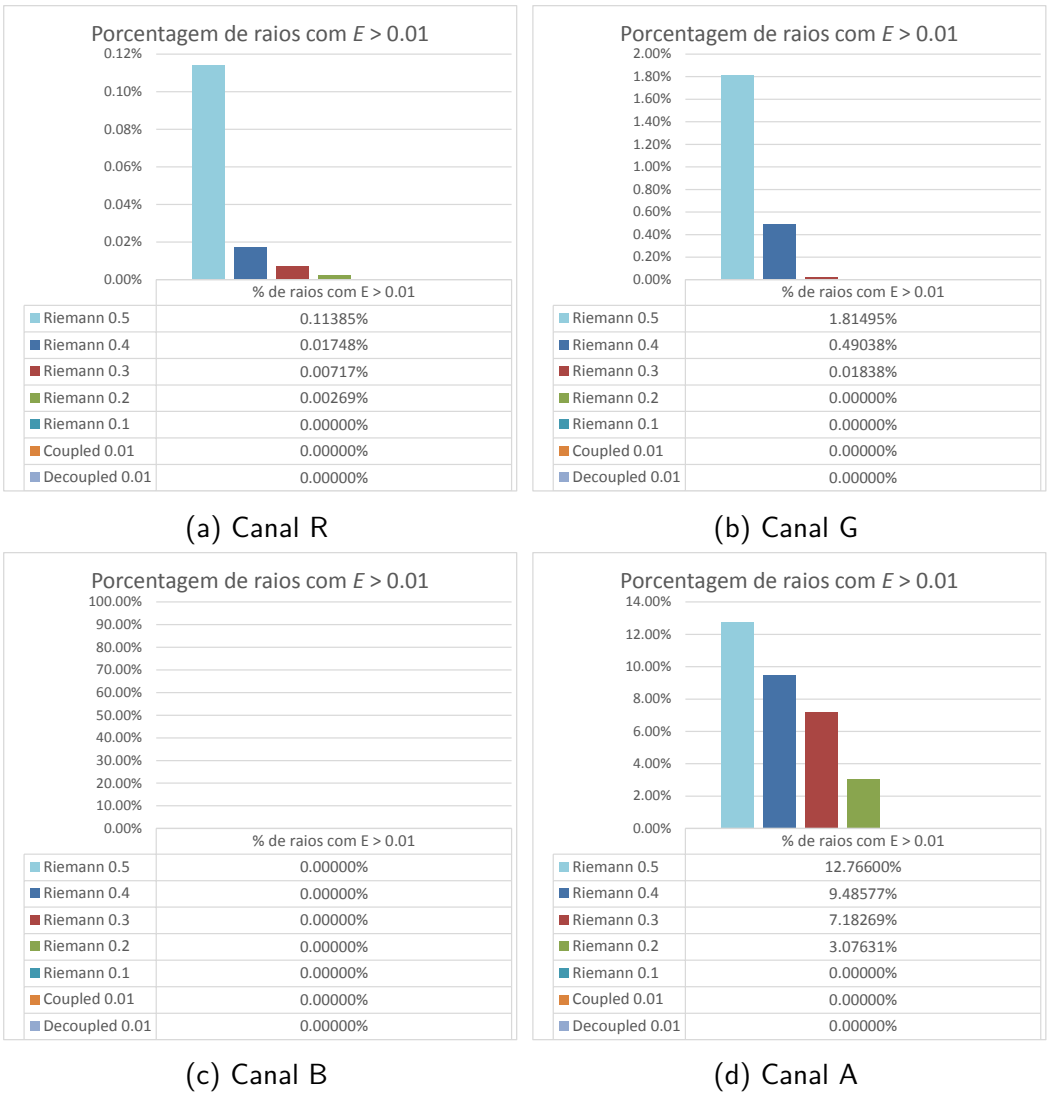
Sabe-se que, na teoria, todos os raios lançados pelo método adaptativo deveriam estar com erro abaixo da tolerância. Porém, além de ter sido determinado um h_{min} para limitar a subdivisão do passo, sabe-se que os

métodos de integração numérica, cuja aproximação normalmente vem da avaliação de amostras, estão sujeitos a gerar erros numéricos na própria avaliação do erro. De fato, os resultados mostram que alguns poucos raios não conseguiram proporcionar um valor de erro exatamente igual ou menor que a tolerância exigida. Porém, a grande maioria dos raios foram avaliados de maneira precisa.

Os testes de precisão foram apresentados em histogramas para cada um dos canais de cor (R, G, B, A), considerando a implementação em CPU. A porcentagem é dada a partir da quantidade total de raios lançados que interceptaram o modelo com a quantidade de raios que geraram um erro maior que a tolerância permitida. As Figuras 5.2, 5.3, 5.4, 5.5, 5.6 e 5.7 mostram os resultados gerados para o modelo Bonsai (Figura 5.1b) e as Figuras 5.8, 5.9, 5.10, 5.11, 5.12 e 5.13 mostram os resultados gerados para o modelo Blunt Fin (Figura 5.1a). Pode-se notar que os resultados dos métodos propostos conseguiram gerar uma boa precisão, mantendo o erro abaixo do permitido para praticamente todos os raios. Também foram adicionados nos histogramas as avaliações feitas utilizando a soma de Riemann, para realizar a comparação dos resultados. Como esperado, o erro diminui de acordo com que o passo diminui. Porém, é difícil conseguir alcançar resultados precisos utilizando um passo de tamanho constante. Além disso, perde-se performance em regiões com pouca variação, que poderiam ter um passo maior, enquanto perde-se precisão e ganha-se um pouco de performance em regiões com alta variação, que poderiam ter um passo menor pra garantir um valor de erro menor que a tolerância definida.

Os resultados de desempenho foram feitos para ambas as implementações em CPU e GPU. Em CPU, foi medido o tempo para renderizar um quadro (em segundos) e em GPU foi medida a taxa de quadros por segundo (*fps*). A Tabela 5.2 representa a média do tempo necessário para gerar um quadro, utilizando cada uma das estratégias, para cada um dos modelos em CPU (a partir do ponto de vista dos modelos da Figura 5.1). Verificou-se que os métodos adaptativos propostos apresentaram um desempenho melhor no geral, principalmente se for considerado o ganho de precisão. Sabe-se que a arquitetura da CPU favorece a implementação de algoritmos adaptativos.

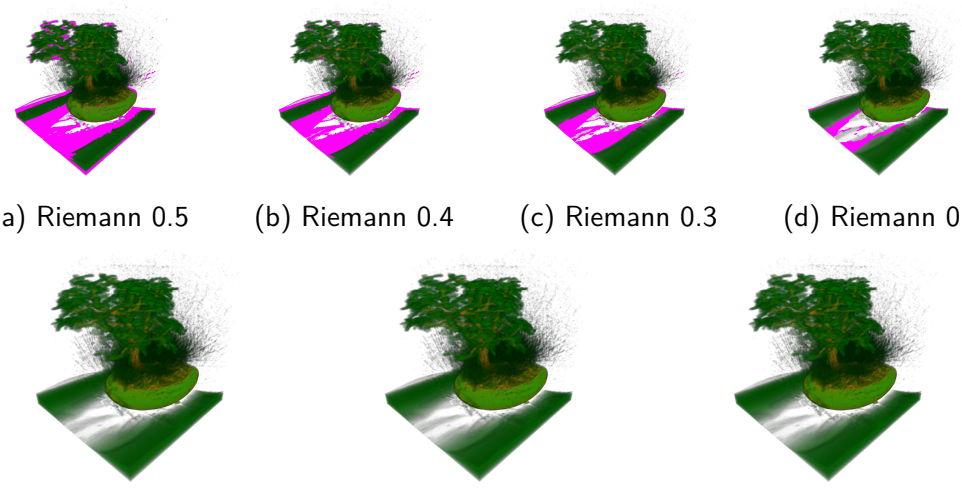
O grande desafio permanece em desenvolver uma implementação do método adaptativo em GPU. Ao contrário do algoritmo em CPU, a arquitetura das placas gráficas (SIMD) acaba favorecendo a implementação que utiliza amostras equidistantes, pois todas as *threads* (sendo uma para cada raio lançado) acabam tendo o mesmo comportamento. No caso do método adaptativo, cada *thread* pode utilizar um tamanho de passo diferente. Dessa forma, acaba



(a) Canal R (b) Canal G

(c) Canal B (d) Canal A

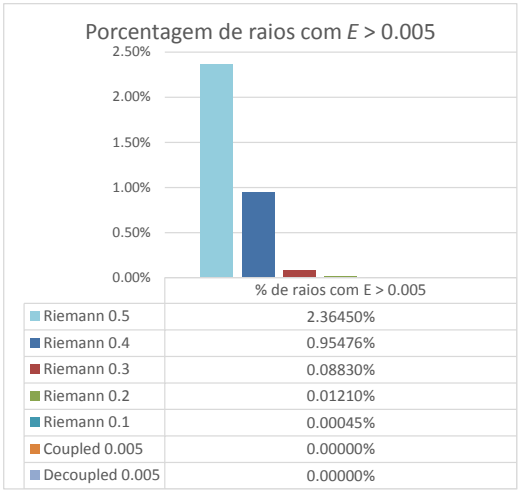
Figura 5.2 – Histogramas do modelo Bonsai com tolerância de 0.01.



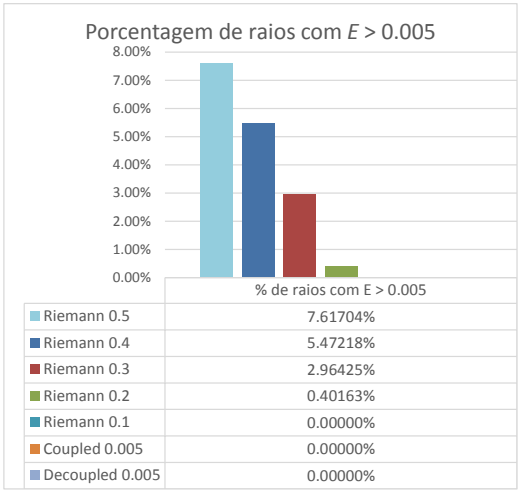
(a) Riemann 0.5 (b) Riemann 0.4 (c) Riemann 0.3 (d) Riemann 0.2

(e) Riemann 0.1 (f) Coupled 0.01 (g) Decoupled 0.01

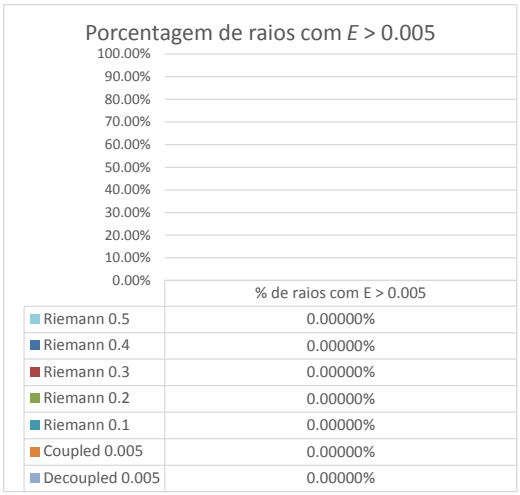
Figura 5.3 – Visualização do modelo Bonsai utilizando cada um dos métodos comparados. Os raios com erro estimado $E > 0.01$ foram ilustrados em magenta.



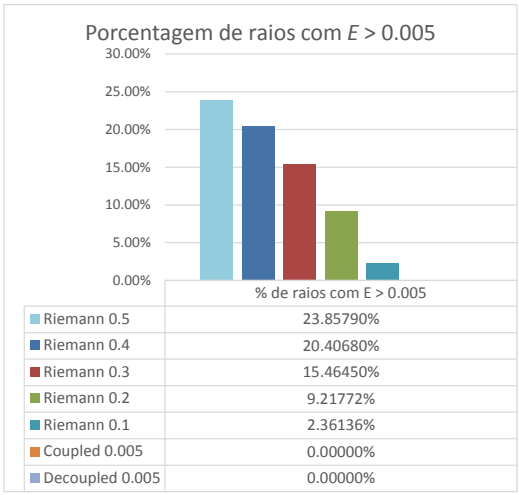
(a) Canal R



(b) Canal G

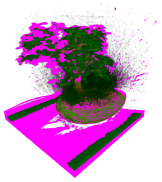


(c) Canal B

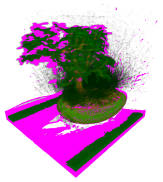


(d) Canal A

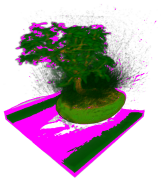
Figura 5.4 – Histogramas do modelo Bonsai com tolerância de 0.005.



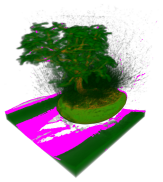
(a) Riemann 0.5



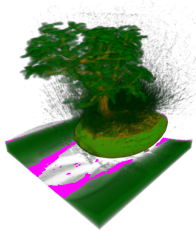
(b) Riemann 0.4



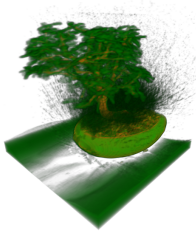
(c) Riemann 0.3



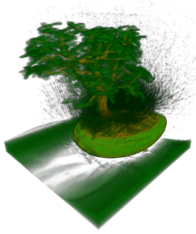
(d) Riemann 0.2



(e) Riemann 0.1

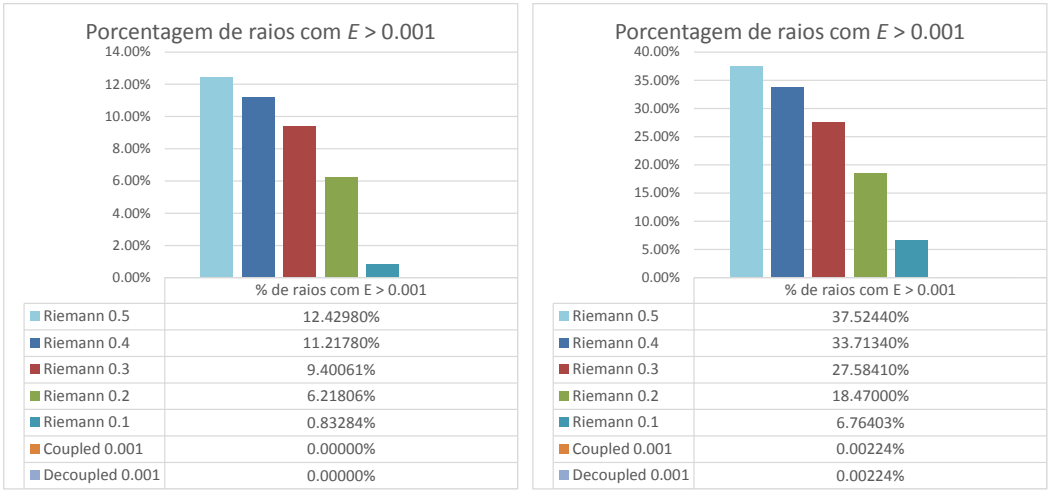


(f) Coupled 0.005



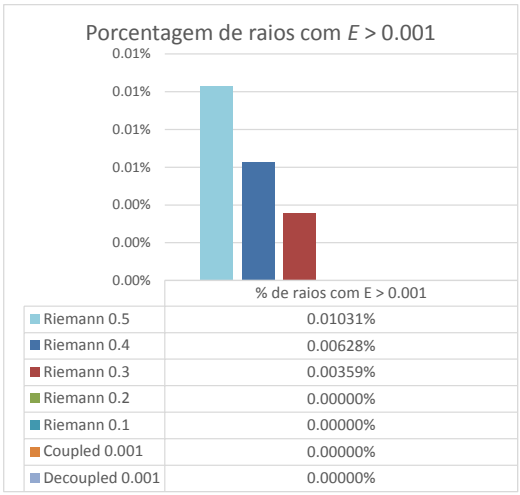
(g) Decoupled 0.005

Figura 5.5 – Visualização do modelo Bonsai utilizando cada um dos métodos comparados. Os raios com erro estimado $E > 0.005$ foram ilustrados em magenta.

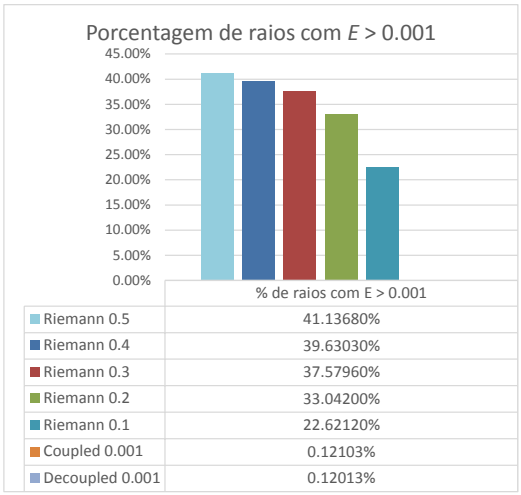


(a) Canal R

(b) Canal G

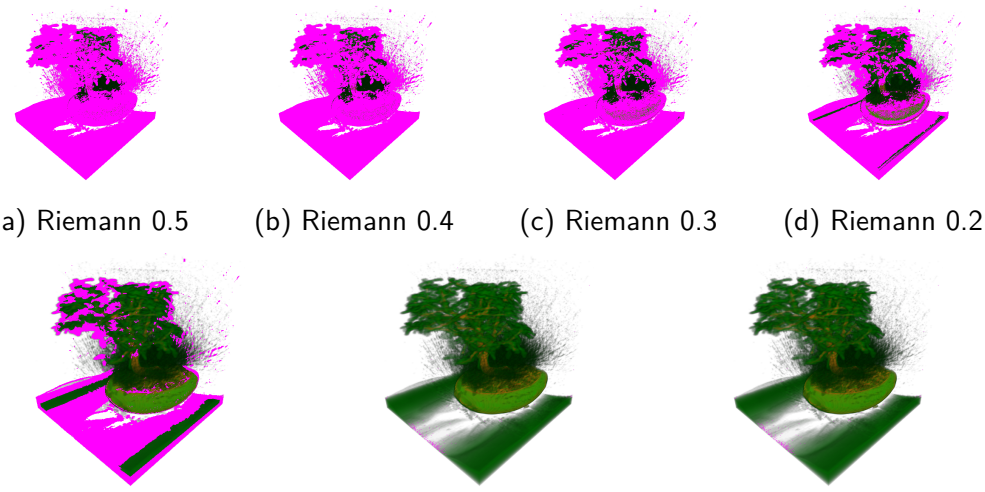


(c) Canal B



(d) Canal A

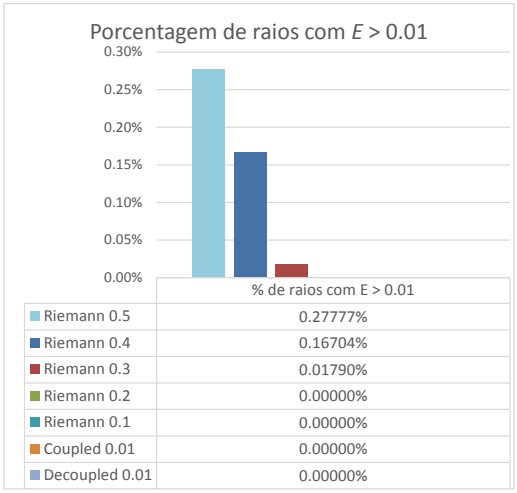
Figura 5.6 – Histogramas do modelo Bonsai com tolerância de 0.001.



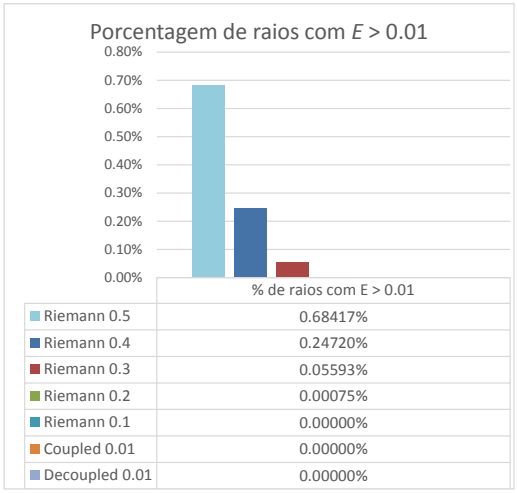
(a) Riemann 0.5 (b) Riemann 0.4 (c) Riemann 0.3 (d) Riemann 0.2

(e) Riemann 0.1 (f) Coupled 0.001 (g) Decoupled 0.001

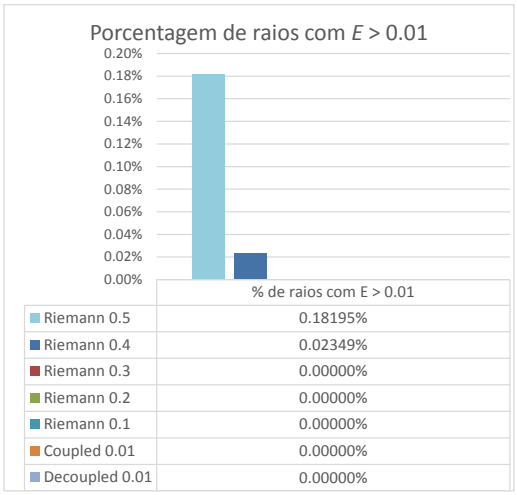
Figura 5.7 – Visualização do modelo Bonsai utilizando cada um dos métodos comparados. Os raios com erro estimado $E > 0.001$ foram ilustrados em magenta.



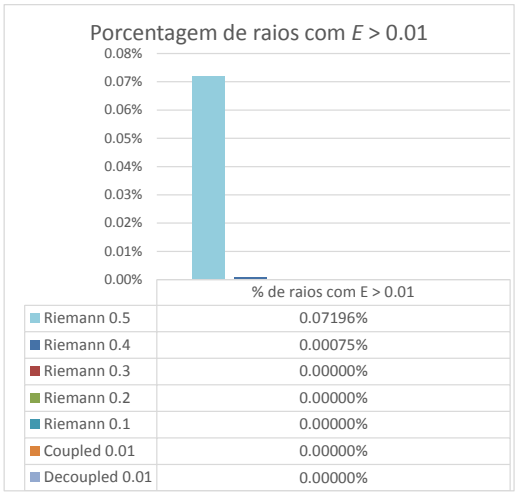
(a) Canal R



(b) Canal G

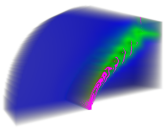


(c) Canal B

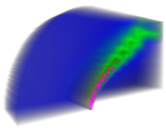


(d) Canal A

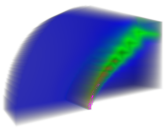
Figura 5.8 – Histogramas do modelo Blunt Fin com tolerância de 0.01.



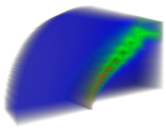
(a) Riemann 0.5



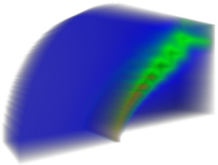
(b) Riemann 0.4



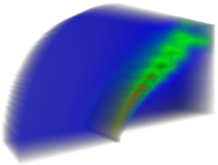
(c) Riemann 0.3



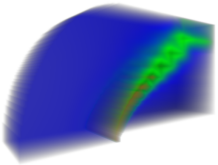
(d) Riemann 0.2



(e) Riemann 0.1

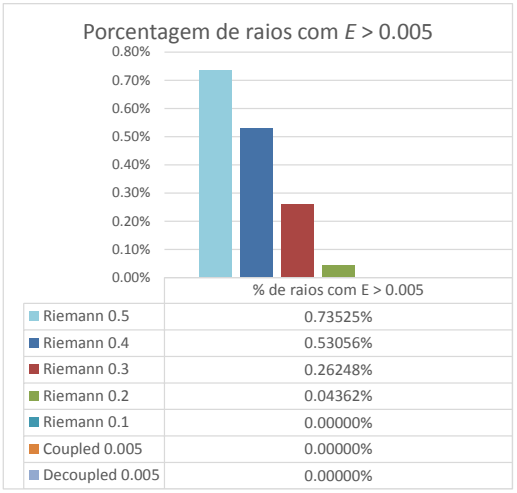


(f) Coupled 0.01

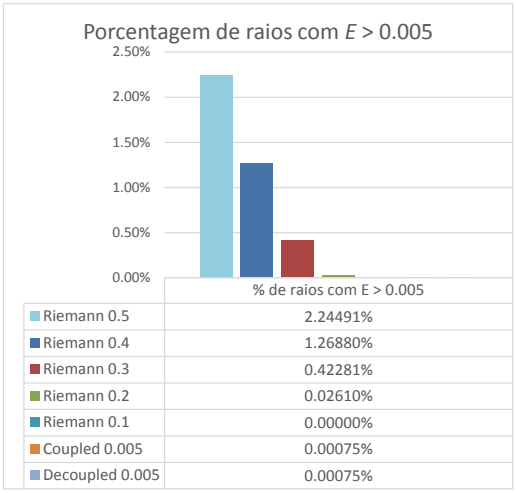


(g) Decoupled 0.01

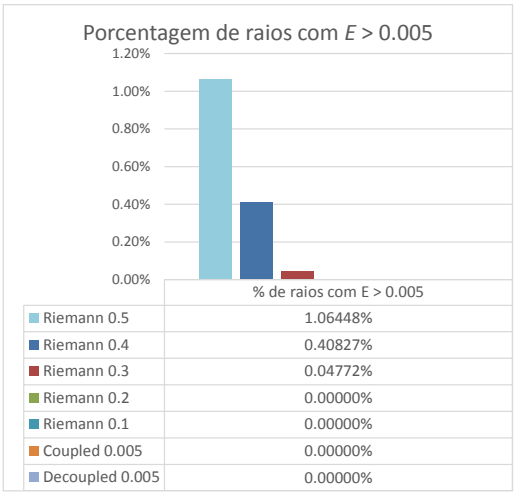
Figura 5.9 – Visualização do modelo Blunt Fin utilizando cada um dos métodos comparados. Os raios com erro estimado $E > 0.01$ foram ilustrados em magenta.



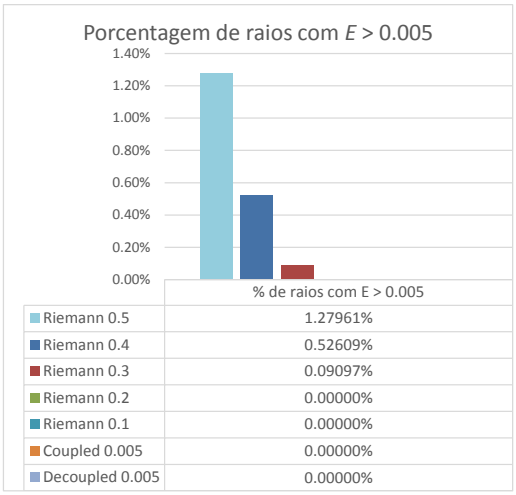
(a) Canal R



(b) Canal G

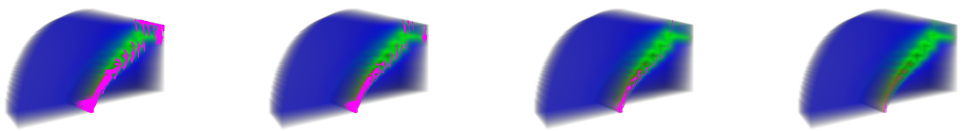


(c) Canal B

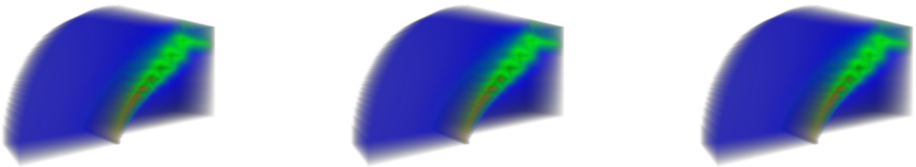


(d) Canal A

Figura 5.10 – Histogramas do modelo Blunt Fin com tolerância de 0.005.

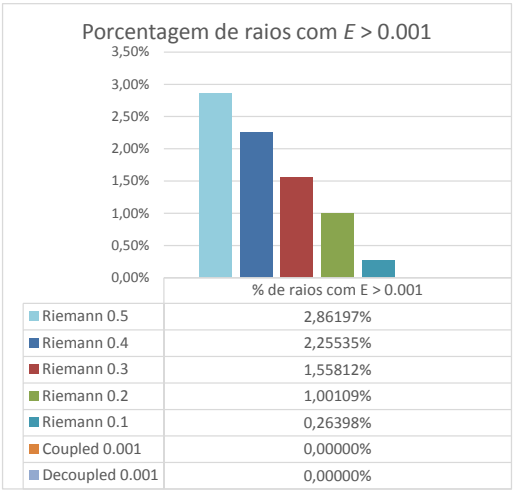


(a) Riemann 0.5 (b) Riemann 0.4 (c) Riemann 0.3 (d) Riemann 0.2

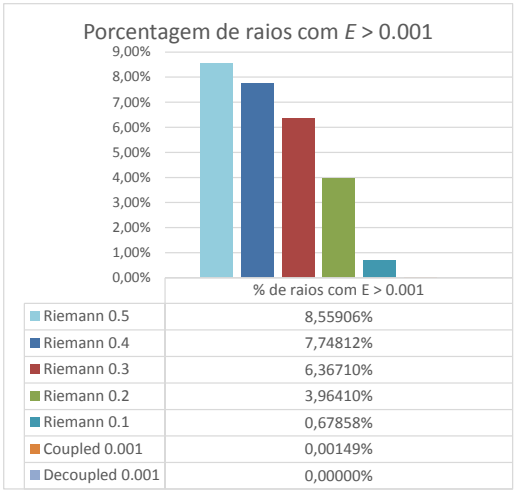


(e) Riemann 0.1 (f) Coupled 0.005 (g) Decoupled 0.005

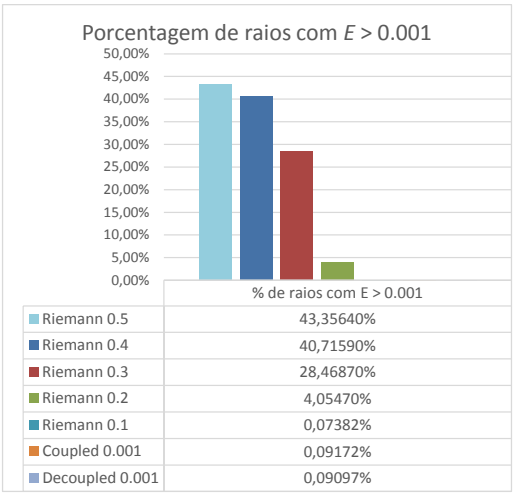
Figura 5.11 – Visualização do modelo Blunt Fin utilizando cada um dos métodos comparados. Os raios com erro estimado $E > 0.005$ foram ilustrados em magenta.



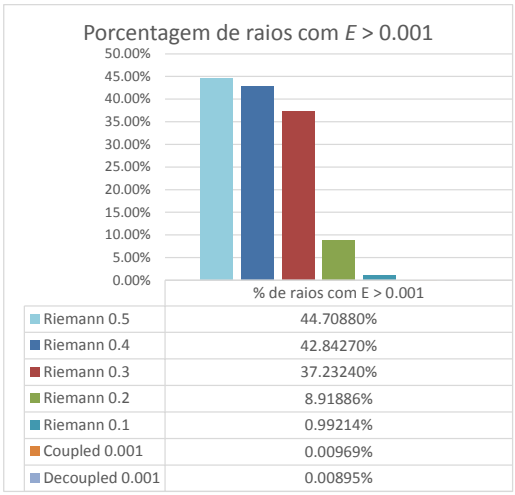
(a) Canal R



(b) Canal G



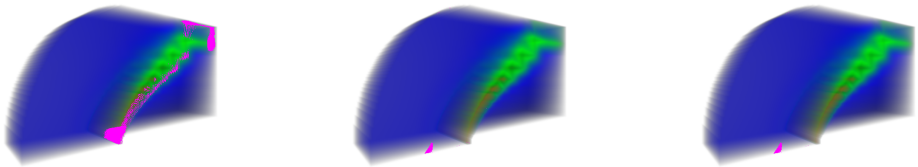
(c) Canal B



(d) Canal A

Figura 5.12 – Histogramas do modelo Blunt Fin com tolerância de 0.001.

(a) Riemann 0.5 (b) Riemann 0.4 (c) Riemann 0.3 (d) Riemann 0.2



(e) Riemann 0.1 (f) Coupled 0.001 (g) Decoupled 0.001

Figura 5.13 – Visualização do modelo Blunt Fin utilizando cada um dos métodos comparados. Os raios com erro estimado $E > 0.001$ foram ilustrados em magenta.

Tabela 5.2 – Testes de performance em CPU (tempo em segundos para gerar um quadro).

	Blunt Fin	Bonsai	Boston Teapot	Engine
Riemann ($h = 0.5$)	6.53	16.23	17.97	8.86
Riemann ($h = 0.4$)	8.05	20.03	21.77	10.91
Riemann ($h = 0.3$)	10.52	26.31	27.87	14.33
Riemann ($h = 0.2$)	15.47	38.94	40.17	21.23
Riemann ($h = 0.1$)	30.33	76.80	76.55	41.70
Coupled ($\epsilon = 0.01$)	9.05	20.03	17.22	9.36
Decoupled ($\epsilon = 0.01$)	9.22	20.14	17.25	9.45
Coupled ($\epsilon = 0.005$)	9.30	23.61	18.61	10.53
Decoupled ($\epsilon = 0.005$)	9.55	23.72	18.66	10.67
Coupled ($\epsilon = 0.001$)	10.94	38.19	23.33	15.25
Decoupled ($\epsilon = 0.001$)	11.28	38.30	23.39	15.34

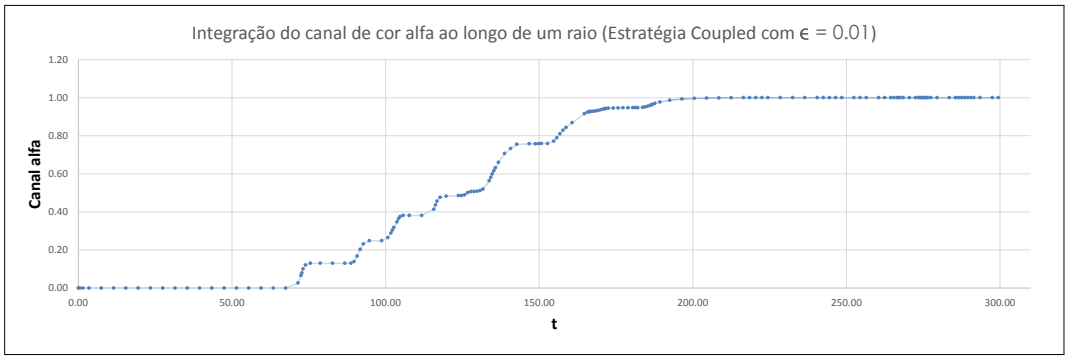
Tabela 5.3 – Testes de performance em GPU (expresso em *fps*).

	Blunt Fin	Bonsai	Boston Teapot	Engine
Riemann ($h = 0.5$)	603	68	149	156
Riemann ($h = 0.4$)	494	55	120	126
Riemann ($h = 0.3$)	390	42	91	96
Riemann ($h = 0.2$)	270	28	62	65
Riemann ($h = 0.1$)	143	14	32	34
Coupled ($\epsilon = 0.01$)	415	31	83	81
Decoupled ($\epsilon = 0.01$)	383	31	80	76
Coupled ($\epsilon = 0.005$)	367	23	68	63
Decoupled ($\epsilon = 0.005$)	335	22	66	59
Coupled ($\epsilon = 0.001$)	226	9	34	29
Decoupled ($\epsilon = 0.001$)	205	9	33	28

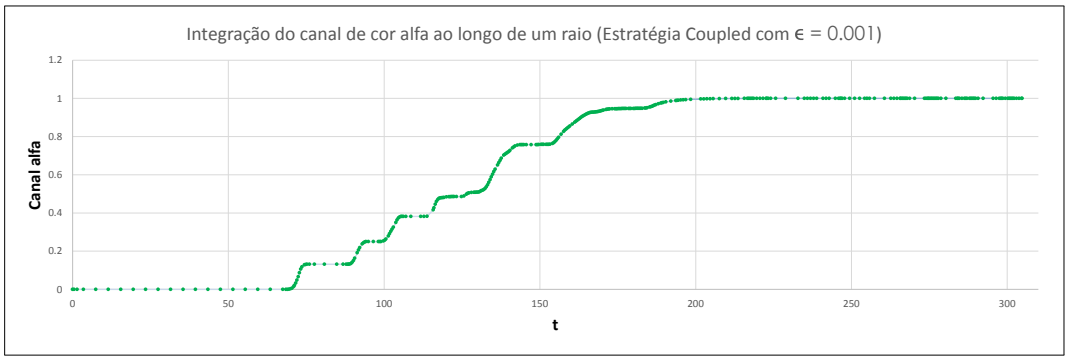
sendo mais difícil conseguir obter todo o poder do processamento paralelo em GPU. A Tabela 5.3 apresenta os resultados gerados em quadros por segundo (*fps*). Pode-se notar que, nesses resultados, a implementação utilizando a soma de Riemann acabou apresentando uma performance superior.

Para testar se o comportamento do passo no método adaptativo estava coerente, foram gerados dois gráficos que ilustram a variação do passo ao longo da avaliação de um raio. Foi escolhido um dos raios lançados no modelo Bonsai, testando o comportamento do tamanho do passo tendo $\epsilon = 0.01$ (Figura 5.14a) e $\epsilon = 0.001$ (Figura 5.14b), com $h_{min} = 0.4$ e $h_{max} = 4.0$. A Figura 5.14 apresenta a avaliação do canal alfa, mostrando as amostras capturadas (marcações na curva) pelo método adaptativo a cada passo de integração feito. Pode-se perceber que o algorimo está de acordo com o esperado, aumentando o tamanho do passo em regiões mais suaves e diminuindo o tamanho do passo em regiões mais íngremes.

Os parâmetros h_{min} e h_{max} afetam diretamente na precisão e performance



(a) Monitoramento do raio com $\epsilon = 0.01$



(b) Monitoramento do raio com $\epsilon = 0.001$

Figura 5.14 – Valor do canal alfa monitorado ao longo de um raio.

Tabela 5.4 – Performance em CPU e GPU para $h_{min} = 0.4$ e $h_{max} = 4.0$.

	CPU (s)	GPU (fps)
Riemann ($h = 0.1$)	76.80	14
Coupled ($\epsilon = 0.001$)	26.77	19
Decoupled ($\epsilon = 0.001$)	26.91	18

do método adaptativo proposto. Além disso, os resultados acima foram gerados a partir de uma configuração global que mantivesse a precisão correta para todos os modelos testados. Porém, a partir da correta manipulação dos parâmetros, podem-se gerar resultados mais competitivos.

Repetiu-se o experimento do modelo Bonsai considerando $\epsilon = 0.001$ e determinando $h_{min} = 0.4$ e $h_{max} = 4.0$. Sabe-se que o método de Simpson adaptativo acaba subdividindo um intervalo em 4 partes iguais. Dessa forma, é esperado que o método de Simpson adaptativo ainda consiga uma precisão maior que a soma de Riemann com passo $h = 0.1$, contando a melhora de performance (principalmente em GPU). A Tabela 5.4 apresenta os resultados de performance gerados para CPU e GPU e as Figuras 5.15, 5.16, 5.17 e 5.18 apresentam os resultados de precisão. Nota-se que essa configuração gerou resultados mais competitivos, mantendo alta precisão.

Como dito anteriormente, sabe-se que parte da perda de performance na GPU em relação à CPU se dá devido às instruções mais complexas com

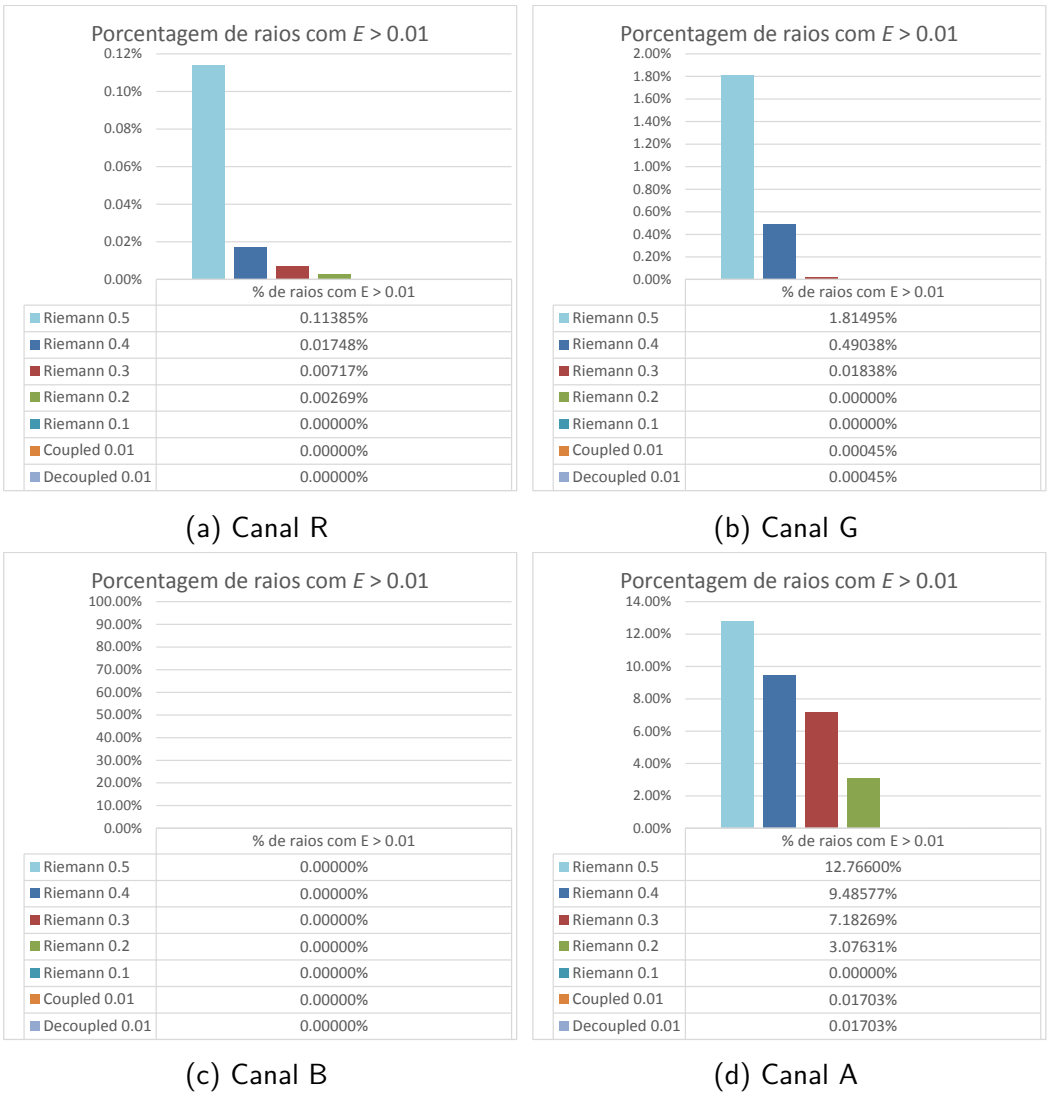
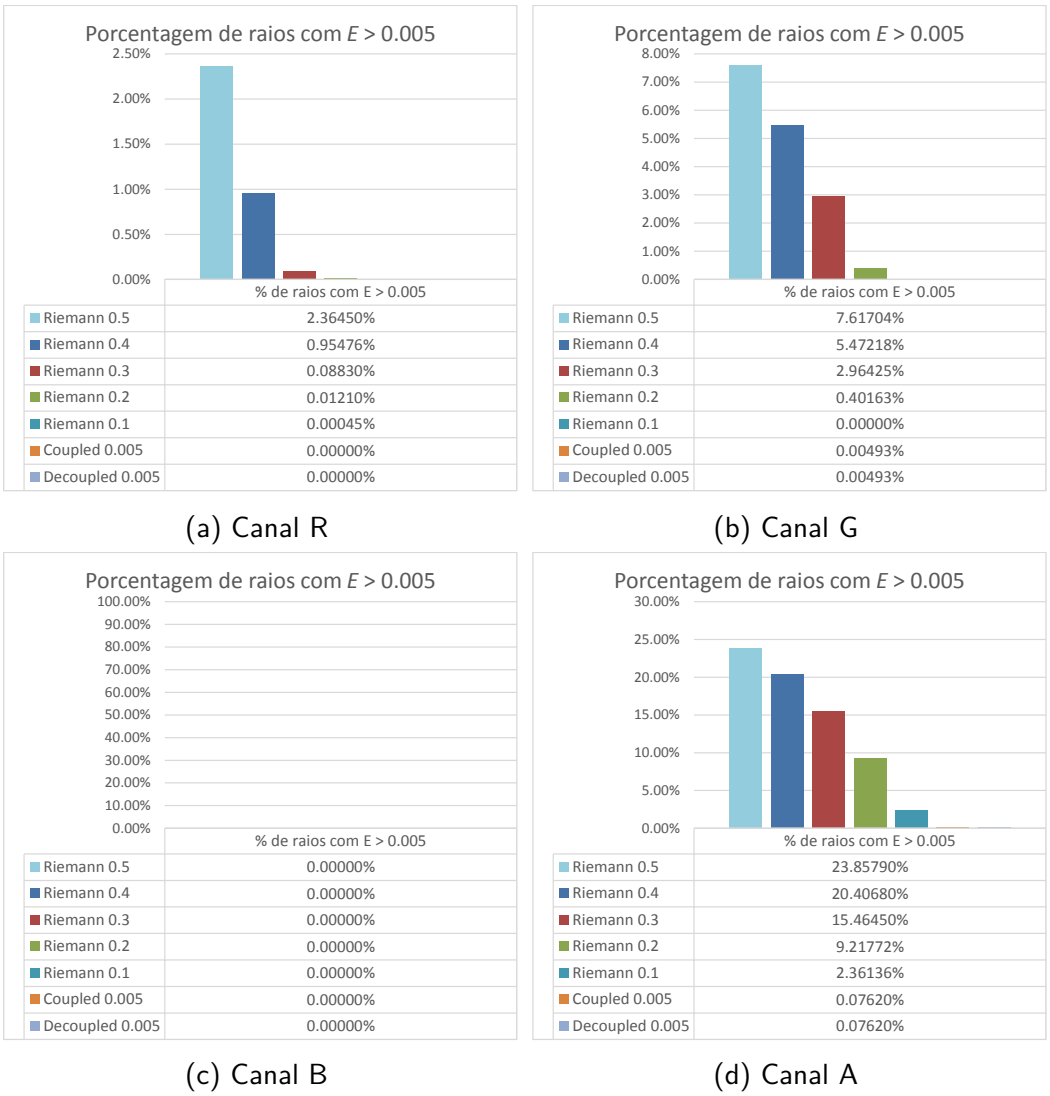


Figura 5.15 – Histogramas do modelo Bonsai com $\epsilon = 0.01$, $h_{max} = 4.0$ e $h_{min} = 0.4$.

grande quantidade de laços de repetição que desfavorecem a paralelização de um algoritmo. Por outro lado, a captura de amostras também está diretamente ligada à performance do método. No caso das implementações feitas, a captura de amostras acaba sendo mais custosa em CPU do que em GPU. Isso ocorre devido à necessidade de identificar a posição e calcular a interpolação trilinear de uma amostra manualmente. No caso das placas gráficas, utilizou-se uma textura 3D para armazenar o dado volumétrico, cuja grande vantagem é que o próprio hardware gráfico gera a interpolação trilinear de forma acelerada ao acessar texturas tridimensionais.

Foi realizado um outro experimento procurando verificar a quantidade total de amostras utilizadas para realizar a visualização de cada um dos modelos. Foram contadas todas as vezes que foi necessário acessar o modelo para capturar o valor de uma amostra, mesmo se ela já tivesse sido calculada em um momento anterior da integração. Este caso pode ocorrer com frequência



(a) Canal R (b) Canal G

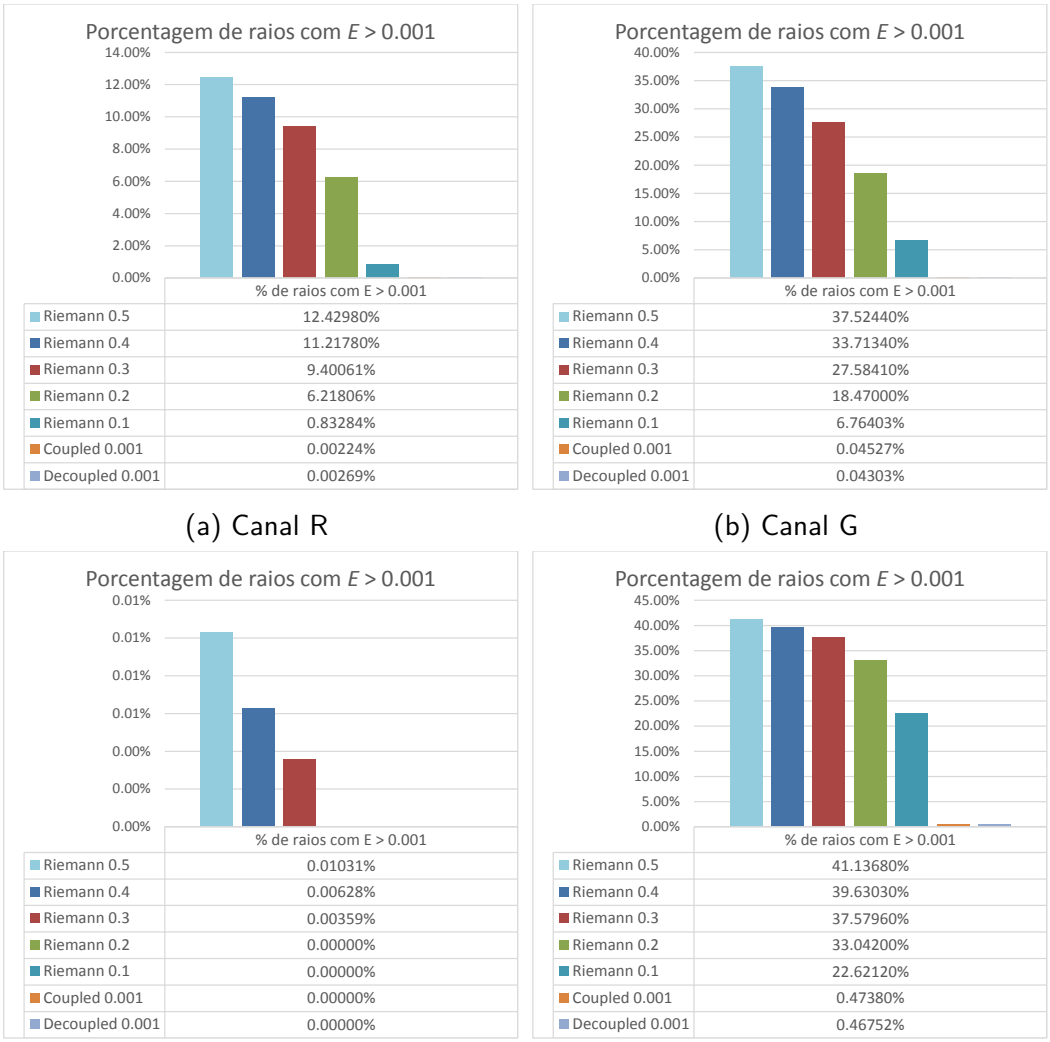
(c) Canal B (d) Canal A

Figura 5.16 – Histogramas do modelo Bonsai com $\epsilon = 0.005$, $h_{max} = 4.0$ e $h_{min} = 0.4$.

nos métodos adaptativos propostos.

As Tabelas 5.5 e 5.6 mostram os resultados com a quantidade total de amostras capturadas. Pode-se perceber que o método adaptativo utilizou, no geral, uma quantidade menor de amostras. Sendo assim, verifica-se que foram obtidos resultados mais precisos utilizando menos amostras, se comparado com a Soma de Riemann.

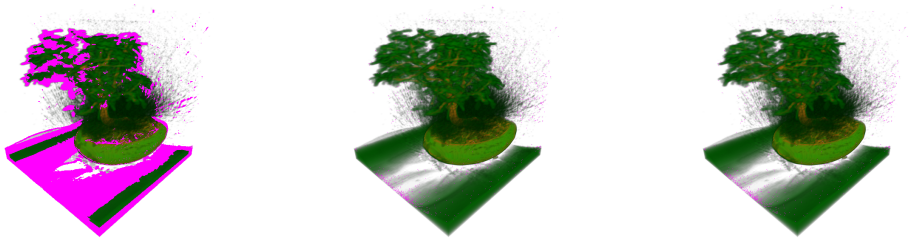
A quantidade de amostras capturadas tem relação direta com a performance dos métodos comparados, principalmente em CPU. Além disso, percebe-se que a estratégia Decoupled apresentou uma quantidade de amostras maior do que a estratégia Coupled.



(a) Canal R (b) Canal G

(c) Canal B (d) Canal A

Figura 5.17 – Histogramas do modelo Bonsai com $\epsilon = 0.001$, $h_{max} = 4.0$ e $h_{min} = 0.4$.



(a) Riemann 0.1 (b) Coupled 0.001 (c) Decoupled 0.001

Figura 5.18 – Visualização do modelo Bonsai dados $h_{min} = 0.4$ e $h_{max} = 4.0$. Os raios com erro estimado $E > 0.001$ foram ilustrados em magenta.

Tabela 5.5 – Quantidade de amostras capturadas em CPU com $h_{min} = 0.1$ e $h_{max} = 2.0$.

	Blunt Fin	Bonsai	B. Teapot	Engine
Riemann ($h = 0.5$)	32346997	73600913	68777689	39273289
Riemann ($h = 0.4$)	40400152	91972993	85938419	49069623
Riemann ($h = 0.3$)	53822739	122593552	114539626	65396760
Riemann ($h = 0.2$)	80666558	183834501	171741693	98050331
Riemann ($h = 0.1$)	161199567	367557701	343348436	196012132
Coupled ($\epsilon = 0.01$)	43321739	103397516	81884281	50147529
Decoupled ($\epsilon = 0.01$)	43964019	103630532	81934915	50602849
Coupled ($\epsilon = 0.005$)	44515321	120477154	88388457	55825175
Decoupled ($\epsilon = 0.005$)	45479747	120815578	88451101	56409035
Coupled ($\epsilon = 0.001$)	52028673	189525554	110941021	78084995
Decoupled ($\epsilon = 0.001$)	53962463	190061450	111028375	78648063

Tabela 5.6 – Quantidade de amostras capturadas em CPU com $h_{min} = 0.4$ e $h_{max} = 4.0$ para modelo Bonsai.

	n° de amostras
Riemann ($h = 0.5$)	73600913
Riemann ($h = 0.4$)	91972993
Riemann ($h = 0.3$)	122593552
Riemann ($h = 0.2$)	183834501
Riemann ($h = 0.1$)	367557701
Coupled ($\epsilon = 0.01$)	72041442
Decoupled ($\epsilon = 0.01$)	72361830
Coupled ($\epsilon = 0.005$)	87639482
Decoupled ($\epsilon = 0.005$)	88010560
Coupled ($\epsilon = 0.001$)	127200736
Decoupled ($\epsilon = 0.001$)	127200736

6
Conclusão

Neste trabalho, foi desenvolvido um método adaptativo para a avaliação da integral volumétrica baseado no método de Simpson adaptativo. Foi discutida a necessidade de utilizar um método adaptativo para que seja possível manter o controle numérico abaixo de uma tolerância pré-determinada, variando o tamanho do passo de acordo com o comportamento da região avaliada.

Buscando contornar a natureza recursiva dos algoritmos adaptativos, foi apresentada uma proposta iterativa baseada no método original. Dessa forma, optou-se por perder um pouco do fator de reutilização de amostras do algoritmo recursivo para possibilitar a implementação em placa gráfica. Foram discutidas duas propostas para controlar o tamanho do passo de integração da integral interna e externa, procurando manter a coerência espacial de ambas em cada fase do processo de avaliação.

Foram revisados alguns métodos numéricos conhecidos, técnicas visualização adaptativa e pré-processamento e trabalhos de análise visando avaliar a corretude e covergência de alguns métodos numéricos utilizados para aproximar a integral volumétrica. A técnica de avaliação adaptativa encontrada na literatura mais semelhante com o método desenvolvido neste trabalho foi a proposta de Novins e Arvo [2]. Eles acabam avaliando o erro apenas em um ponto, através do cálculo da n–ésima derivada (dependendo do método numérico utilizado). Por outro lado, o método utilizado de base no presente trabalho busca capturar algumas amostras ao longo do intervalo que está sendo avaliado para calcular uma estimativa de erro. Dessa forma, o algoritmo, adaptativo e iterativo, proposto se diferencia do método de Novins e Arvo pois, assim como o algoritmo recursivo de Simpson adaptativo utilizado como base, busca reavaliar um intervalo caso a sua respectiva aproximação gere um erro maior do que a tolerância desejada.

Nos resultados gerados, foi visto que o método implementado consegue manter o erro controlado como esperado, tendo em mente as limitações do próprio método numérico em si com o domínio que está sendo trabalhado e contando com o auxílio dos limites de passo mínimo e máximo estipulados. Os testes de precisão foram feitos apenas em CPU. Já em relação ao desempenho, foram obtidos resultados em CPU e GPU. Em CPU, o algoritmo adaptativo mostrou um desempenho competitivo, comparando com a estratégia de Riemann [20] com amostras equidistantes. Além disso, geralmente apresentou o controle de precisão melhor, na medida em que se diminuiu o erro tolerado.

Em GPU, foi visto que é possível encontrar um resultado competitivo a partir do ajuste dos parâmetros adotados (tolerância, passo máximo e mínimo), mantendo boa precisão. De fato, o método proposto, assim como muitos métodos de visualização volumétrica, possuem o *trade-off* entre performance e precisão, variando de acordo com a manipulação dos parâmetros.

Sabe-se que, comparando com a abordagem de amostras equidistantes, o método adaptativo possui instruções mais complexas. Dessa forma, é natural que a proporcionalidade de tempo do método adaptativo nas implementações de CPU e GPU não se mantenham. Além disso, o algoritmo adaptativo não consegue tirar todo o proveito da paralelização uma vez que nem todos os raios lançados avançam da mesma forma (justamente pelo passo ser adaptado ao longo do raio a partir da variação do campo escalar). Mesmo assim, foram obtidos resultados satisfatórios para ambas as implementações (CPU e GPU).

O presente trabalho foi publicado na 28ª edição do SIBGRAPI [38], com o título "Accurate Volume Rendering based on Adaptive Numerical Integration" [39].

6.1 Trabalhos futuros

Como continuação do trabalho, pretende-se estendê-lo para malhas não estruturadas. O algoritmo adaptativo proposto é facilmente estendido em qualquer tipo de domínio, dados os devidos pontos inicial e final da integração e suporte para que seja possível capturar uma amostra em um ponto qualquer. Também é de interesse avaliar a utilização de um método de interpolação diferente da interpolação trilinear [40, 41], que foi utilizada neste trabalho.

Outro tópico de interesse que não foi discutido neste trabalho são algoritmos de iluminação e métricas de qualidade visual [42] para saber quando é necessário realizar uma avaliação mais cuidadosa da integral volumétrica. Por fim, algumas questões que foram consideradas em aberto: Como ou quanto a tolerância da integral interna afeta a imagem final? A diferença abosluta entre dois valores de cor, utilizando o espaço de cor RGB mais o alfa, é a melhor forma de avaliar o erro?

Referências Bibliográficas

[1] MAX, N.. **Optical models for direct volume rendering**. IEEE Transactions on Visualization and Computer Graphics, 1(2):99–108, June 1995.

[2] NOVINS, K.; ARVO, J.. **Controlled precision volume integration**. In: PROCEEDINGS OF THE 1992 WORKSHOP ON VOLUME VISUALIZATION, VVS '92, p. 83–89, New York, NY, USA, 1992. ACM.

[3] MARCHESIN, S.; DE VERDIERE, G.. **High-quality, semi-analytical volume rendering for amr data**. IEEE Transactions on Visualization and Computer Graphics, 15(6):1611–1618, Nov 2009.

[4] LINDHOLM, S.; JÖNSSON, D.; KNUTSSON, H. ; YNNERMAN, A.. **Towards data centric sampling for volume rendering**. In: SIGRAD 2013, 2013.

[5] MIRANDA, F.; CELES, W.. **Volume rendering of unstructured hexahedral meshes**. The Visual Computer, 28(10):1005–1014, 2012.

[6] KAUFMAN, A. E.. **Volume visualization**. ACM Comput. Surv., 28(1):165–167, Mar. 1996.

[7] DREBIN, R. A.; CARPENTER, L. ; HANRAHAN, P.. **Volume rendering**. In: PROCEEDINGS OF THE 15TH ANNUAL CONFERENCE ON COMPUTER GRAPHICS AND INTERACTIVE TECHNIQUES, SIGGRAPH '88, p. 65–74, New York, NY, USA, 1988. ACM.

[8] LEVOY, M.. **Volume rendering: display of surfaces from volume data**. Computer Graphics and Applications, 1988.

[9] KINDLMANN, G.; DURKIN, J. W.. **Semi-automatic generation of transfer functions for direct volume rendering**. In: PROCEEDINGS OF THE 1998 IEEE SYMPOSIUM ON VOLUME VISUALIZATION, VVS '98, p. 79–86, New York, NY, USA, 1998. ACM.

[10] KUNCIR, G. F.. **Algorithm 103: Simpson's rule integrator**. Commun. ACM, 5(6):347–, June 1962.

[11] MCKEEMAN, W. M.. **Algorithm 145: Adaptive numerical integration by simpson's rule**. Commun. ACM, 5(12):604–, Dec. 1962.

[12] LYNESS, J. N.. **Notes on the adaptive simpson quadrature routine**. J. ACM, 16(3):483–495, July 1969.

[13] ELVINS, T. T.. **A survey of algorithms for volume visualization**. SIGGRAPH Comput. Graph., 26(3):194–201, Aug. 1992.

[14] GARRITY, M. P.. **Raytracing irregular volume data**. In: PROCEEDINGS OF THE 1990 WORKSHOP ON VOLUME VISUALIZATION, VVS '90, p. 35–40, New York, NY, USA, 1990. ACM.

[15] KAUFMAN, A. E.; MUELLER, K.. **Overview of volume rendering**. In: THE VISUALIZATION HANDBOOK, p. 127–174. Elsevier, 2005.

[16] BLINN, J. F.. **Light reflection functions for simulation of clouds and dusty surfaces**. SIGGRAPH Comput. Graph., 16(3):21–29, July 1982.

[17] WILLIAMS, P. L.; MAX, N.. **A volume density optical model**. In: PROCEEDINGS OF THE 1992 WORKSHOP ON VOLUME VISUALIZATION, VVS '92, p. 61–68, New York, NY, USA, 1992. ACM.

[18] HADWIGER, M.; KNISS, J. M.; REZK-SALAMA, C.; WEISKOPF, D. ; ENGEL, K.. **Real-time Volume Graphics**. A. K. Peters, Ltd., Natick, MA, USA, 2006.

[19] ENGEL, K.; KRAUS, M. ; ERTL, T.. **High-quality pre-integrated volume rendering using hardware-accelerated pixel shading**. In: PROCEEDINGS OF THE ACM SIGGRAPH/EUROGRAPHICS WORKSHOP ON GRAPHICS HARDWARE, HWWS '01, p. 9–16, New York, NY, USA, 2001. ACM.

[20] ETIENE, T.; JONSSON, D.; ROPINSKI, T.; SCHEIDEGGER, C.; COMBA, J.; NONATO, L.; KIRBY, R.; YNNERMAN, A. ; SILVA, C.. **Verifying volume rendering using discretization error analysis**. IEEE Transactions on Visualization and Computer Graphics, 20(1):140–154, Jan 2014.

[21] ETIENE, T.; KIRBY, R. M. ; SILVA, C. T.. **A study of discretization errors in volume rendering integral approximations**.

[22] PFISTER, H.; LORENSEN, B.; BAJAJ, C.; KINDLMANN, G.; SCHROEDER, W.; AVILA, L. S.; MARTIN, K.; MACHIRAJU, R. ; LEE, J.. **The transfer function bake-off**. IEEE Comput. Graph. Appl., 21(3):16–22, May 2001.

[23] WILLIAMS, P.; MAX, N. ; STEIN, C.. **A high accuracy volume renderer for unstructured data**. IEEE Transactions on Visualization and Computer Graphics, 4(1):37–54, Jan 1998.

[24] NGUYEN, B. P.; TAY, W.-L.; CHUI, C.-K. ; ONG, S.-H.. **Automatic transfer function design for volumetric data visualization using clustering on lh space**. Proceedings of Computer Graphics International, p. 1–10, 2011.

[25] ZHOU, J.; TAKATSUKA, M.. **Automatic transfer function generation using contour tree controlled residue flow model and color harmonics**. IEEE Transactions on Visualization and Computer Graphics, 15(6):1481–1488, Nov. 2009.

[26] EL HAJJAR, J.-F.; MARCHESIN, S.; DISCHLER, J.-M. ; MONGENET, C.. **Second order pre-integrated volume rendering**. In: VISUALIZATION SYMPOSIUM, 2008. PACIFICVIS '08. IEEE PACIFIC, p. 9–16, March 2008.

[27] NOVINS, K.; ARVO, J. ; SALESIN, D.. **Adaptive error bracketing for controlled-precision volume rendering**. Technical report, 1992.

[28] WANG, H.; XIAO, L. ; CAO, Y.. **An adaptive sampling based parallel volume rendering algorithm**. In: VIRTUAL REALITY AND VISUALIZATION (ICVRV), 2011 INTERNATIONAL CONFERENCE ON, p. 158–163, Nov 2011.

[29] LORENSEN, W. E.; CLINE, H. E.. **Marching cubes: A high resolution 3d surface construction algorithm**. SIGGRAPH Comput. Graph., 21(4):163–169, Aug. 1987.

[30] MORELAND, K.; ANGEL, E.. **A fast high accuracy volume renderer for unstructured data**. In: PROCEEDINGS OF THE 2004 IEEE SYMPOSIUM ON VOLUME VISUALIZATION AND GRAPHICS, VV '04, p. 9–16, Washington, DC, USA, 2004. IEEE Computer Society.

[31] MAX, N.; BECKER, B. ; CRAWFIS, R.. **Flow volumes for interactive vector field visualization**. In: PROCEEDINGS OF THE 4TH CONFERENCE ON VISUALIZATION '93, VIS '93, p. 19–24, Washington, DC, USA, 1993. IEEE Computer Society.

[32] RÖTTGER, S.; KRAUS, M. ; ERTL, T.. **Hardware-accelerated volume and isosurface rendering based on cell-projection**. In: PROCEEDINGS OF THE CONFERENCE ON VISUALIZATION '00, VIS '00, p. 109–116, Los Alamitos, CA, USA, 2000. IEEE Computer Society Press.

[33] LUM, E. B.; WILSON, B. ; MA, K.-L.. **High-quality lighting and efficient pre-integration for volume rendering**. In: PROCEEDINGS

OF THE SIXTH JOINT EUROGRAPHICS - IEEE TCVG CONFERENCE ON VISUALIZATION, VISSYM'04, p. 25–34, Aire-la-Ville, Switzerland, Switzerland, 2004. Eurographics Association.

[34] ROY, C. J.. **Review of code and solution verification procedures for computational simulation**. J. Comput. Phys., 205(1):131–156, May 2005.

[35] CARR, H.; MOLLER, T. ; SNOEYINK, J.. **Artifacts caused by simplicial subdivision**. IEEE Transactions on Visualization and Computer Graphics, 12(2):231–242, Mar. 2006.

[36] **volvis.org**. `http://volvis.org/`.

[37] **Nasa advanced supercomputing division**. `http://www.nas.nasa.gov/publications/datasets.html`. Accessed: 2014-11-15.

[38] **Sibgrapi 2015 - xxviii conference on graphics, patterns and images**. http://sibgrapi2015.dcc.ufba.br/.

[39] CAMPAGNOLO, L. Q.; CELES, W. ; FIGUEIREDO, L. H. D.. **Accurate volume rendering based on adaptive numerical integration**. In: Papa, J. P.; Sander, P. V.; Marroquim, R. G. ; Farrell, R., editors, PROCEEDINGS..., Los Alamitos, 2015. Conference on Graphics, Patterns and Images, 28. (SIBGRAPI), IEEE Computer SocietyÂ´s Conference Publishing Services.

[40] NEUMANN, L.; CSÉBFALVI, B.; KÖNIG, A. ; GRÖLLER, E.. **Gradient estimation in volume data using 4d linear regression**, 2000.

[41] HONG, D.; NING, G.; ZHAO, T.; ZHANG, M. ; ZHENG, X.. **Method of normal estimation based on approximation for visualization**. Journal of Electronic Imaging, 12(3):470–477, 2003.

[42] MENEGHEL, G. B.; NETTO, M. L.. **A comparison of global illumination methods using perceptual quality metrics**. In: Papa, J. P.; Sander, P. V.; Marroquim, R. G. ; Farrell, R., editors, PROCEEDINGS..., Los Alamitos, 2015. Conference on Graphics, Patterns and Images, 28. (SIBGRAPI), IEEE Computer SocietyÂ´s Conference Publishing Services.

A
Algoritmo recursivo de Simpson Adaptativo

Algorithm 5 Função principal do Simpson Adaptativo.

Input: f, a, b, ϵ

Output: $\int(a, b, \epsilon)$

1: $c = \frac{a+b}{2}$, $h = b - a$
2: $fa = f(a)$, $fb = f(b)$, $fc = f(c)$
3: $S = (\frac{h}{6})(fa + 4fc + fb)$
4: $Sr = \int(f, a, b, \epsilon, S, fa, fb, fc)$ {Algoritmo 6}
5: **return** Sr

Algorithm 6 Função auxiliar (recursão) do método de Simpson Adaptativo.

Input: $f, a, b, \epsilon, S, fa, fb, fc$

Output: $\int(a, b, \epsilon)$

1: $c = \frac{a+b}{2}$, $h = b - a$
2: $d = \frac{a+c}{2}$, $e = \frac{c+b}{2}$
3: $fd = f(d)$, $fe = f(e)$
4: $Sleft = (\frac{h}{12})(fa + 4fd + fc)$
5: $Sright = (\frac{h}{12})(fc + 4fe + fb)$
6: $S2 = Sleft + Sright$
7: **if** $|S2 - S| \leq 15\epsilon$ **then**
8: $\quad Sr = S2 + \frac{S2-S}{15}$
9: **else**
10: $\quad Sr = \int(f, a, c, \frac{\epsilon}{2}, Sleft, fa, fc, fd) +$
$\int(f, c, b, \frac{\epsilon}{2}, Sright, fc, fb, fe)$ {Algoritmo 6}
11: **end if**
12: **return** Sr

B
Resultados gerados

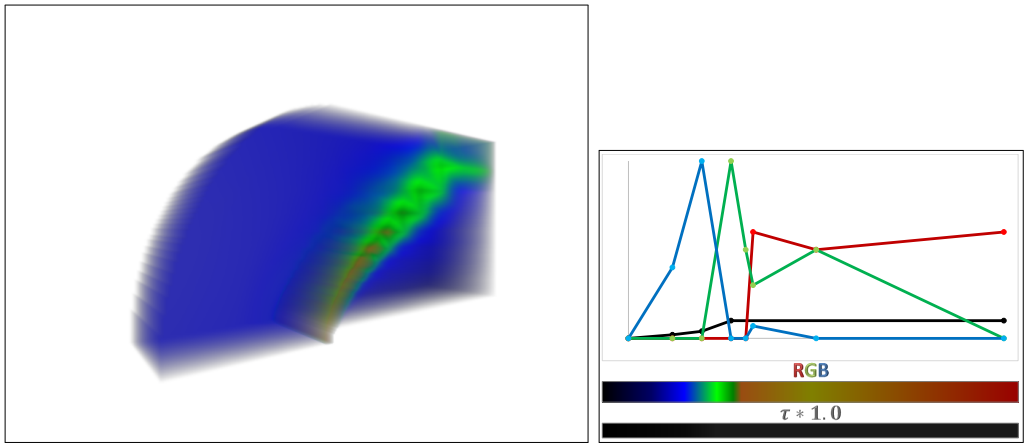


Figura B.1 – Modelo Blunt Fin e a respectiva função de transferência utilizada.

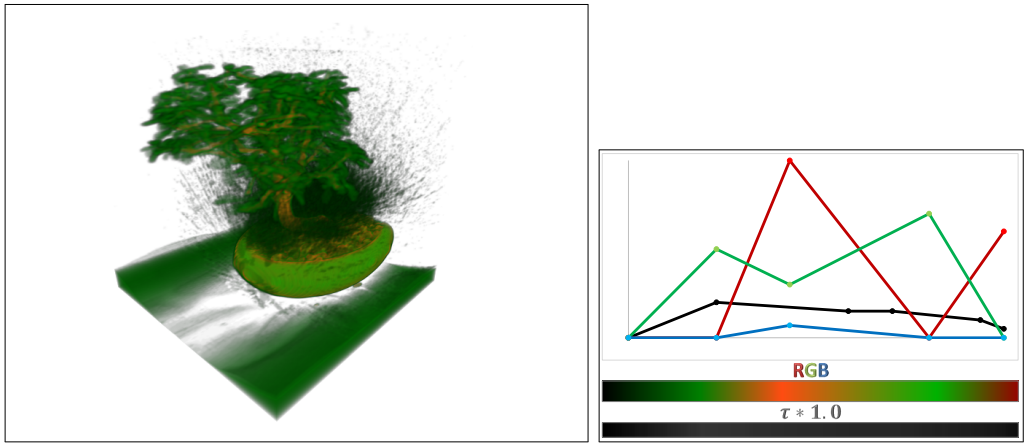


Figura B.2 – Modelo Bonsai e a respectiva função de transferência utilizada.

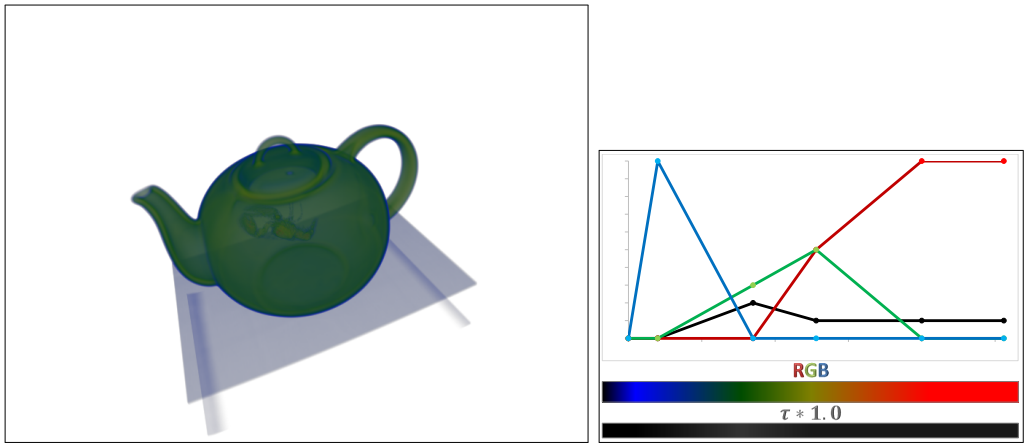


Figura B.3 – Modelo Boston Teapot e a respectiva função de transferência utilizada.

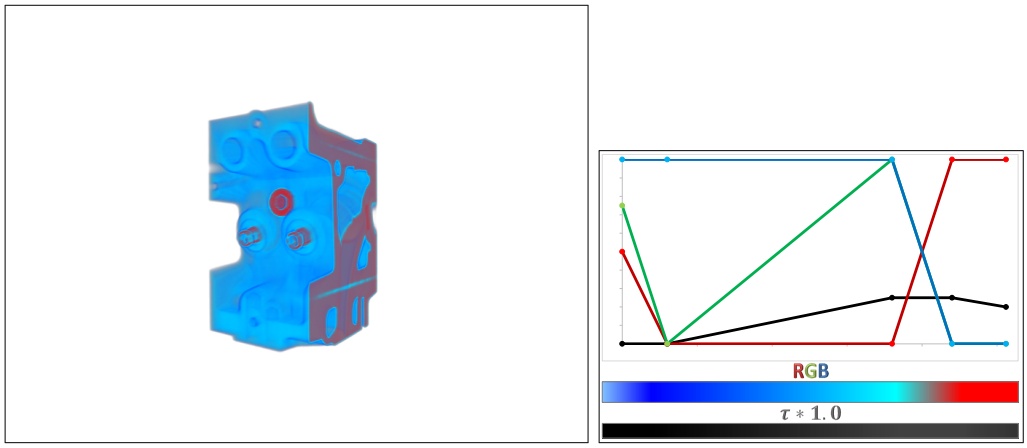


Figura B.4 – Modelo Engine e a respectiva função de transferência utilizada.

Tabela B.1 – Porcentagem de raios com erro acima da tolerância permitida.

Modelo	Tolerância		$R_{h=0.5}$	$R_{h=0.4}$	$R_{h=0.3}$	$R_{h=0.2}$	$R_{h=0.1}$	Coupled	Decoupled
Blunt Fin	$E > 0.01$	R	0.2778	0.1670	0.0179	0.0000	0.0000	0.0000	0.0000
		G	0.6842	0.2472	0.0559	0.0008	0.0000	0.0000	0.0000
		B	0.1820	0.0235	0.0000	0.0000	0.0000	0.0000	0.0000
		A	0.0720	0.0008	0.0000	0.0000	0.0000	0.0000	0.0000
	$E > 0.005$	R	0.7352	0.5306	0.2625	0.0436	0.0000	0.0000	0.0000
		G	2.2449	1.2688	0.4228	0.0262	0.0000	0.0008	0.0008
		B	1.0645	0.4083	0.0477	0.0000	0.0000	0.0000	0.0000
		A	1.2796	0.5261	0.0910	0.0000	0.0000	0.0000	0.0000
	$E > 0.001$	R	2.8620	2.2553	1.5581	1.0011	0.2640	0.0000	0.0000
		G	8.5591	7.7481	6.3671	3.9641	0.6786	0.0015	0.0000
		B	43.3564	40.7159	28.4687	4.0547	0.0738	0.0917	0.0910
		A	44.7088	42.8427	37.2324	8.9189	0.9921	0.0097	0.0090
Bonsai	$E > 0.01$	R	0.1139	0.0175	0.0072	0.0027	0.0000	0.0000	0.0000
		G	1.8150	0.4904	0.0184	0.0000	0.0000	0.0000	0.0000
		B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
		A	12.7660	9.4858	7.1827	3.0763	0.0000	0.0000	0.0000
	$E > 0.005$	R	2.3645	0.9548	0.0883	0.0121	0.0004	0.0000	0.0000
		G	7.6170	5.4722	2.9643	0.4016	0.0000	0.0000	0.0000
		B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
		A	23.8579	20.4068	15.4645	9.2177	2.3614	0.0000	0.0000
	$E > 0.001$	R	12.4298	11.2178	9.4006	6.2181	0.8328	0.0000	0.0000
		G	37.5244	33.7134	27.5841	18.4700	6.7640	0.0022	0.0022
		B	0.0103	0.0063	0.0036	0.0000	0.0000	0.0000	0.0000
		A	41.1368	39.6303	37.5796	33.0420	22.6212	0.1210	0.1201
Boston Teapot	$E > 0.01$	R	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
		G	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
		B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
		A	1.2636	0.2369	0.0000	0.0000	0.0000	0.0000	0.0000
	$E > 0.005$	R	0.0115	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
		G	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
		B	0.1044	0.0078	0.0000	0.0000	0.0000	0.0007	0.0007
		A	22.0318	6.7123	2.9178	0.1625	0.0000	0.0030	0.0030
	$E > 0.001$	R	8.4398	5.3217	2.2940	0.9978	0.0000	0.0000	0.0000
		G	16.5779	15.0414	12.5860	4.7849	0.0000	0.0000	0.0000
		B	28.2702	25.7148	21.9025	16.3757	0.0018	0.0430	0.0430
		A	33.6841	33.0239	31.9806	29.5549	18.1217	0.0526	0.0515
Engine	$E > 0.01$	R	4.1196	2.2585	0.0896	0.0000	0.0000	0.0000	0.0000
		G	5.1547	2.1762	0.1765	0.0000	0.0000	0.0000	0.0000
		B	9.2077	5.5640	1.6102	0.5435	0.0000	0.0000	0.0000
		A	6.9543	4.5875	1.9715	0.5435	0.0000	0.0000	0.0000
	$E > 0.005$	R	7.5107	6.2044	4.3084	0.4132	0.0000	0.0000	0.0000
		G	13.3284	10.1876	5.1716	1.0176	0.0000	0.0000	0.0000
		B	16.1828	13.9621	10.4689	3.5738	0.3794	0.0000	0.0000
		A	11.1945	10.0765	8.2606	4.1585	0.3794	0.0000	0.0000
	$E > 0.001$	R	11.3597	10.9182	10.8619	10.4723	8.2955	0.0006	0.0000
		G	28.5111	26.1573	22.3602	17.6757	9.4299	0.0000	0.0000
		B	26.1686	25.0974	23.6203	21.3482	15.8869	0.0011	0.0006
		A	18.8992	18.0766	16.7235	14.7142	10.8303	0.0006	0.0006

Tabela B.2 – Testes de performance em CPU (tempo em segundos para gerar um quadro).

	Blunt Fin	Bonsai	Boston Teapot	Engine
Riemann ($h = 0.5$)	6.53	16.23	17.97	8.86
Riemann ($h = 0.4$)	8.05	20.03	21.77	10.91
Riemann ($h = 0.3$)	10.52	26.31	27.87	14.33
Riemann ($h = 0.2$)	15.47	38.94	40.17	21.23
Riemann ($h = 0.1$)	30.33	76.80	76.55	41.70
Coupled ($\epsilon = 0.01$)	9.05	20.03	17.22	9.36
Decoupled ($\epsilon = 0.01$)	9.22	20.14	17.25	9.45
Coupled ($\epsilon = 0.005$)	9.30	23.61	18.61	10.53
Decoupled ($\epsilon = 0.005$)	9.55	23.72	18.66	10.67
Coupled ($\epsilon = 0.001$)	10.94	38.19	23.33	15.25
Decoupled ($\epsilon = 0.001$)	11.28	38.30	23.39	15.34

Tabela B.3 – Tabela de performance em GPU (quadros por segundo).

	Blunt Fin	Bonsai	Boston Teapot	Engine
Riemann ($h = 0.5$)	603	68	149	156
Riemann ($h = 0.4$)	494	55	120	126
Riemann ($h = 0.3$)	390	42	91	96
Riemann ($h = 0.2$)	270	28	62	65
Riemann ($h = 0.1$)	143	14	32	34
Coupled ($\epsilon = 0.01$)	415	31	83	81
Decoupled ($\epsilon = 0.01$)	383	31	80	76
Coupled ($\epsilon = 0.005$)	367	23	68	63
Decoupled ($\epsilon = 0.005$)	335	22	66	59
Coupled ($\epsilon = 0.001$)	226	9	34	29
Decoupled ($\epsilon = 0.001$)	205	9	33	28

Tabela B.4 – Erros RGBA para o modelo Bonsai com $h_{min} = 0.4$ e $h_{max} = 4.0$.

	$E > 0.001$			
	R	G	B	A
Riemann ($h = 0.1$)	0.8328	6.7640	0.0000	22.6212
Coupled	0.0022	0.0453	0.0000	0.4738
Decoupled	0.0027	0.0430	0.0000	0.4675

Tabela B.5 – Performance em CPU e GPU para o modelo Bonsai com $h_{min} = 0.4$ e $h_{max} = 4.0$.

	CPU (s)	GPU (fps)
Riemann ($h = 0.1$)	76.80	14
Coupled ($\epsilon = 0.001$)	26.77	19
Decoupled ($\epsilon = 0.001$)	26.91	18

C
Resultados adicionais

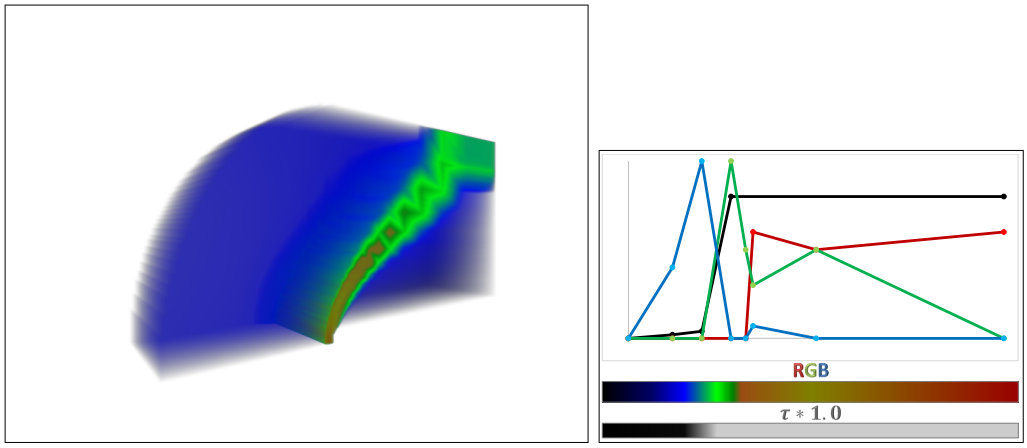


Figura C.1 – Modelo Blunt Fin e a respectiva função de transferência utilizada.

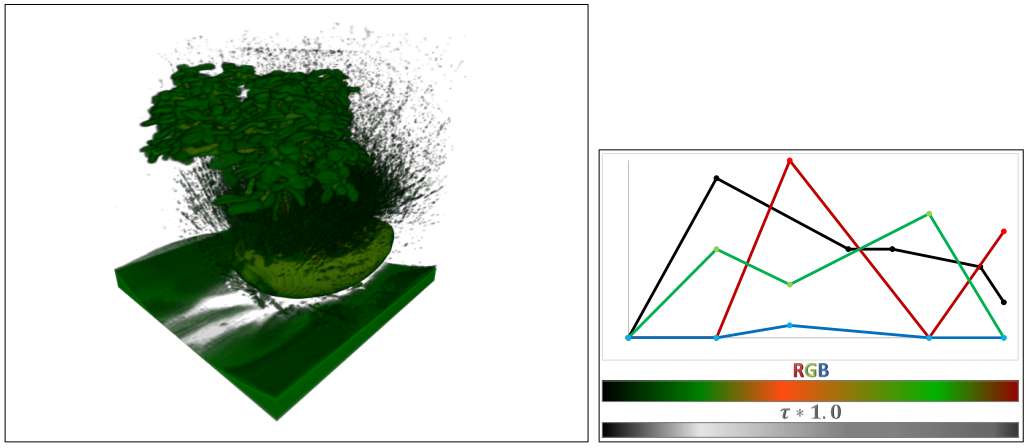


Figura C.2 – Modelo Bonsai e a respectiva função de transferência utilizada.

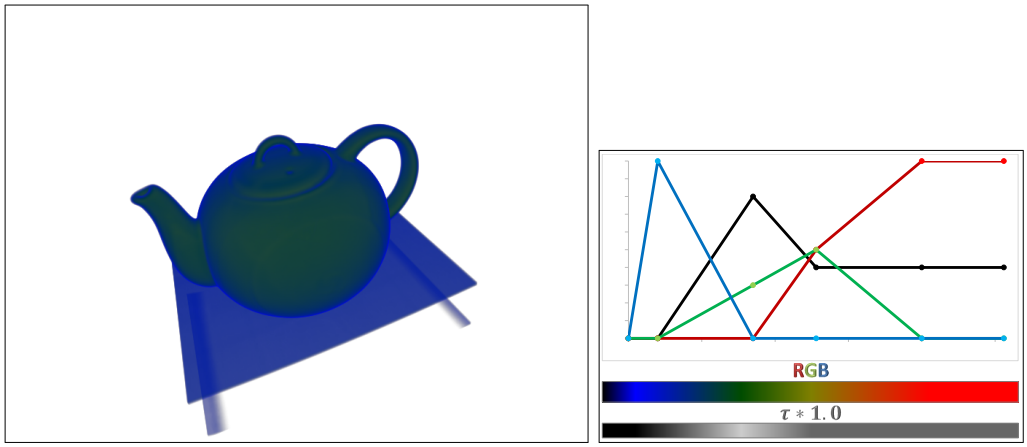


Figura C.3 – Modelo Boston Teapot e a respectiva função de transferência utilizada.

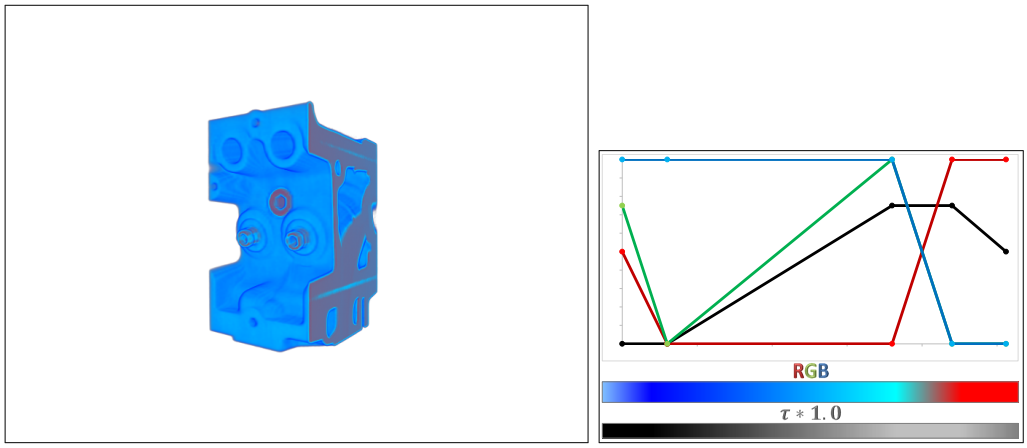


Figura C.4 – Modelo Engine e a respectiva função de transferência utilizada.

Tabela C.1 – Porcentagem de raios com erro acima da tolerância permitida.

Modelo	Tolerância		$R_{h=0.5}$	$R_{h=0.4}$	$R_{h=0.3}$	$R_{h=0.2}$	$R_{h=0.1}$	Coupled	Decoupled
Blunt Fin	$E > 0.01$	R	2.6744	2.3068	1.7706	0.8896	0.0392	0.0000	0.0000
		G	6.0032	4.6017	2.7333	0.8680	0.0168	0.0000	0.0000
		B	6.4245	4.5204	1.7199	0.1834	0.0000	0.0004	0.0004
		A	0.5634	0.4694	0.3494	0.2207	0.0276	0.0004	0.0004
	$E > 0.005$	R	3.5003	3.2896	2.9101	2.2005	0.5361	0.0000	0.0000
		G	9.7842	8.0427	5.7444	3.1092	0.4262	0.0000	0.0000
		B	10.8241	8.9595	5.8063	2.5324	0.0250	0.0004	0.0004
		A	1.5659	0.8206	0.5988	0.4467	0.2069	0.0011	0.0011
	$E > 0.001$	R	4.2803	4.1945	4.1162	3.9734	3.4455	0.0351	0.0339
		G	15.0261	14.5578	13.6577	12.0023	8.0367	0.0328	0.0309
		B	45.4298	43.5030	34.0319	14.4661	8.7220	0.0917	0.0910
		A	36.3372	35.5140	32.1770	9.0766	1.2300	0.0578	0.0570
Bonsai	$E > 0.01$	R	4.7563	4.0158	2.8432	1.0220	0.0000	0.0000	0.0000
		G	5.5121	4.0028	2.0816	0.3581	0.0000	0.0000	0.0000
		B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
		A	12.1735	10.3809	8.3091	6.2665	2.7832	0.0031	0.0031
	$E > 0.005$	R	6.4574	5.9670	4.9405	3.5425	0.6908	0.0000	0.0000
		G	18.1755	12.2559	7.2862	3.7657	0.2577	0.0000	0.0000
		B	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
		A	17.0141	15.6326	13.5393	10.1976	6.0446	0.0188	0.0188
	$E > 0.001$	R	9.3706	9.0501	8.4786	7.6614	5.9926	0.0005	0.0005
		G	46.3540	43.8281	39.9234	33.1935	15.2865	0.0067	0.0067
		B	3.6536	2.7836	1.4061	0.1587	0.0000	0.0000	0.0000
		A	25.2219	24.2066	22.9049	20.8851	16.7016	3.4466	3.4475
Boston Teapot	$E > 0.01$	R	11.4046	4.2217	0.5483	0.0000	0.0000	0.0000	0.0000
		G	5.2547	0.3499	0.0000	0.0000	0.0000	0.0000	0.0000
		B	31.7192	24.3390	9.5464	0.0000	0.0000	0.0063	0.0059
		A	14.3172	13.4287	12.3265	0.3003	0.0000	0.0122	0.0118
	$E > 0.005$	R	18.4127	17.5064	14.4779	2.5853	0.0000	0.0000	0.0000
		G	20.7463	18.2717	10.0414	0.0811	0.0000	0.0000	0.0000
		B	36.0176	35.3127	33.5126	20.7793	0.0000	0.0218	0.0218
		A	15.2014	15.0070	14.7215	13.2880	0.1322	0.0315	0.0311
	$E > 0.001$	R	20.6852	20.5705	20.3350	19.8663	18.0477	0.0000	0.0000
		G	36.5678	36.1757	35.3701	29.4864	19.3672	0.0000	0.0000
		B	37.9110	37.8414	37.7599	37.5330	35.9736	0.0352	0.0348
		A	16.1488	16.0018	15.8126	15.5375	15.0914	0.0381	0.0374
Engine	$E > 0.01$	R	10.0089	9.7749	9.4687	8.8751	0.0039	0.0000	0.0000
		G	14.5028	10.0156	9.4169	6.7569	0.0000	0.0000	0.0000
		B	12.5318	12.0373	11.3788	10.1402	0.2796	0.0000	0.0000
		A	2.6475	2.3487	1.8977	1.1112	0.2757	0.0000	0.0000
	$E > 0.005$	R	10.3747	10.1988	10.0179	9.7089	8.6980	0.0000	0.0000
		G	32.2901	29.2203	20.7117	10.3347	5.4721	0.0000	0.0000
		B	13.6723	13.2963	12.7962	11.9911	9.8662	0.0000	0.0000
		A	3.7198	3.4239	2.9458	2.3397	0.9911	0.0000	0.0000
	$E > 0.001$	R	10.8399	10.7480	10.6769	10.5084	10.1701	0.0941	0.0947
		G	36.9831	36.8128	36.4373	35.5178	31.3644	0.0006	0.0006
		B	16.5589	16.1372	15.6230	14.9369	13.5353	0.4719	0.4713
		A	6.8652	6.4463	5.8825	5.1180	3.7458	0.3681	0.3676

Tabela C.2 – Testes de performance em CPU (tempo em segundos para gerar um quadro).

	Blunt Fin	Bonsai	Boston Teapot	Engine
Riemann ($h = 0.5$)	6.53	16.23	17.97	8.86
Riemann ($h = 0.4$)	8.05	20.03	21.77	10.91
Riemann ($h = 0.3$)	10.52	26.31	27.87	14.33
Riemann ($h = 0.2$)	15.47	38.94	40.17	21.23
Riemann ($h = 0.1$)	30.33	76.80	76.55	41.70
Coupled ($\epsilon = 0.01$)	9.30	29.94	18.48	11.36
Decoupled ($\epsilon = 0.01$)	9.25	29.95	18.50	11.39
Coupled ($\epsilon = 0.005$)	9.76	37.05	20.69	13.36
Decoupled ($\epsilon = 0.005$)	9.84	37.06	20.67	13.53
Coupled ($\epsilon = 0.001$)	12.39	55.44	26.44	19.31
Decoupled ($\epsilon = 0.001$)	12.61	55.47	26.45	19.31

Tabela C.3 – Tabela de performance em GPU (quadros por segundo).

	Blunt Fin	Bonsai	Boston Teapot	Engine
Riemann ($h = 0.5$)	603	68	149	156
Riemann ($h = 0.4$)	494	55	120	126
Riemann ($h = 0.3$)	390	42	91	96
Riemann ($h = 0.2$)	270	28	62	65
Riemann ($h = 0.1$)	143	14	32	34
Coupled ($\epsilon = 0.01$)	329	15	55	56
Decoupled ($\epsilon = 0.01$)	306	14	53	53
Coupled ($\epsilon = 0.005$)	289	10	39	39
Decoupled ($\epsilon = 0.005$)	266	9	38	38
Coupled ($\epsilon = 0.001$)	166	6	23	22
Decoupled ($\epsilon = 0.001$)	152	6	22	21

Tabela C.4 – Erros RGBA para o modelo Bonsai com $h_{min} = 0.4$ e $h_{max} = 4.0$.

	$E > 0.001$			
	R	G	B	A
Riemann ($h = 0.1$)	5.9926	15.2865	0.0000	16.7016
Coupled	0.3958	0.1457	0.0000	3.9065
Decoupled	0.3945	0.1448	0.0000	3.9042

Tabela C.5 – Performance em CPU e GPU para o modelo Bonsai com $h_{min} = 0.4$ e $h_{max} = 4.0$.

	CPU (s)	GPU (fps)
Riemann ($h = 0.1$)	76.80	14
Coupled ($\epsilon = 0.001$)	32.61	18
Decoupled ($\epsilon = 0.001$)	33.53	17

D
Resultados adicionais com outras funções de transferência

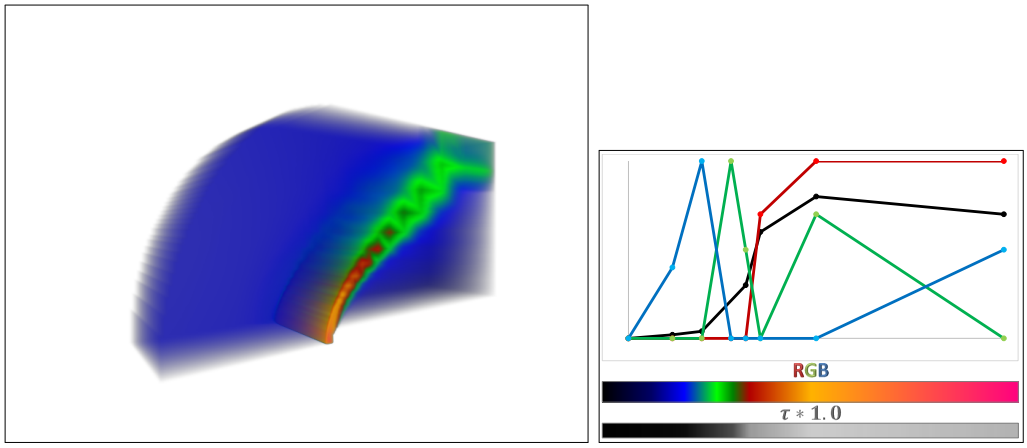


Figura D.1 – Modelo Blunt Fin e a respectiva função de transferência utilizada.

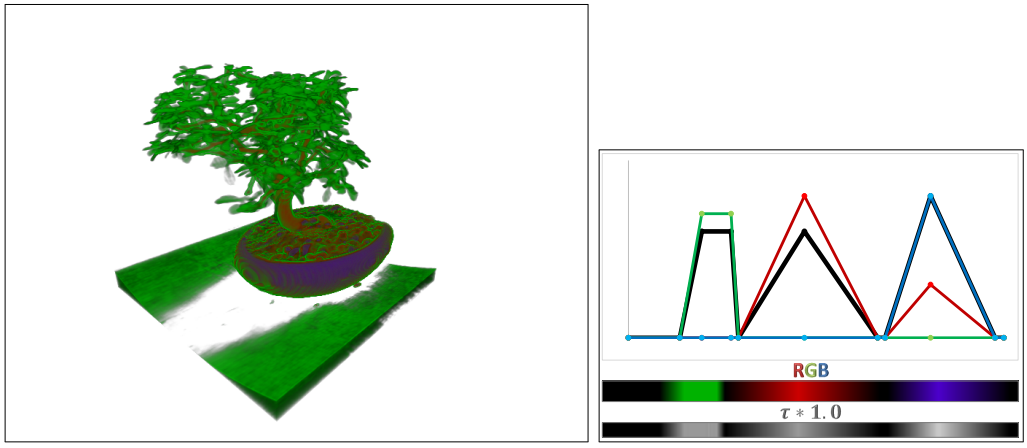


Figura D.2 – Modelo Bonsai e a respectiva função de transferência utilizada.

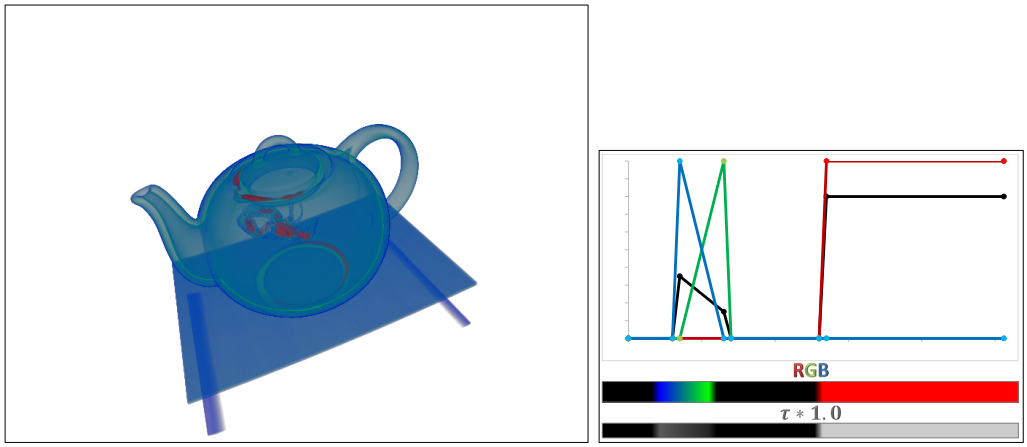


Figura D.3 – Modelo Boston Teapot e a respectiva função de transferência utilizada.

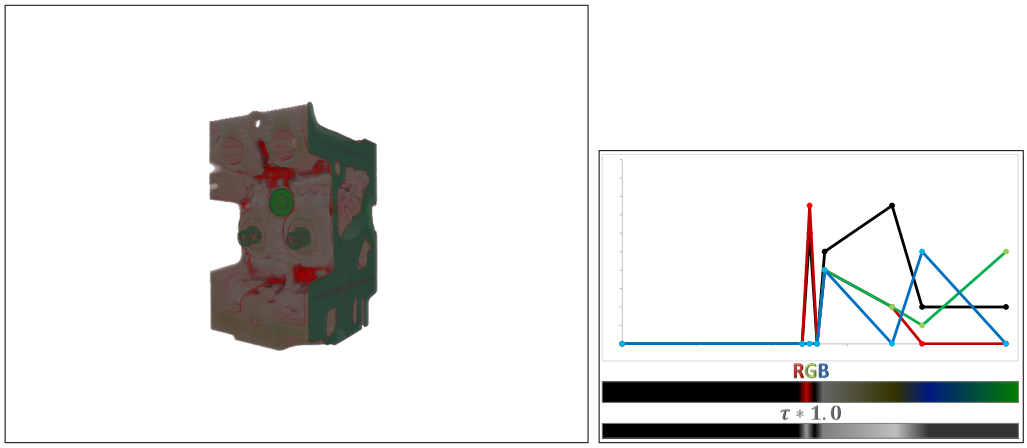


Figura D.4 – Modelo Engine e a respectiva função de transferência utilizada.

Tabela D.1 – Porcentagem de raios com erro acima da tolerância permitida.

Modelo	Tolerância		$R_{h=0.5}$	$R_{h=0.4}$	$R_{h=0.3}$	$R_{h=0.2}$	$R_{h=0.1}$	Coupled	Decoupled
Blunt Fin	$E > 0.01$	R	2.3747	1.7818	1.0015	0.1793	0.0138	0.0000	0.0000
		G	2.6975	1.7240	0.5201	0.1521	0.0015	0.0000	0.0000
		B	1.1491	0.3706	0.0582	0.0000	0.0000	0.0000	0.0000
		A	0.5361	0.2979	0.1115	0.0686	0.0149	0.0000	0.0000
	$E > 0.005$	R	3.7762	3.4350	2.8217	1.6241	0.1324	0.0000	0.0000
		G	5.3690	4.2959	2.4943	0.8628	0.0392	0.0000	0.0000
		B	4.2057	2.0480	0.7371	0.1010	0.0000	0.0000	0.0000
		A	1.3154	1.0846	0.7274	0.2379	0.0623	0.0000	0.0000
	$E > 0.001$	R	4.9249	4.8470	4.7087	4.3992	3.6509	0.0142	0.0142
		G	12.0702	11.4546	10.3498	8.7522	4.1856	0.0022	0.0022
		B	44.4918	42.2506	31.3060	8.6802	2.0432	0.0917	0.0910
		A	39.3468	38.2235	34.1356	9.1250	1.2132	0.0276	0.0269
Bonsai	$E > 0.01$	R	8.6597	7.1195	5.1553	1.3264	0.0000	0.0005	0.0005
		G	22.0048	17.9325	13.1798	8.0088	1.0027	0.0148	0.0148
		B	4.0006	3.4905	2.7746	1.7517	0.0063	0.0005	0.0005
		A	15.0485	13.4496	10.9560	6.9482	2.2878	0.0278	0.0278
	$E > 0.005$	R	11.1801	10.1680	8.9568	6.2275	0.3138	0.0009	0.0009
		G	32.3544	29.1234	23.7180	16.6640	6.3557	0.0175	0.0175
		B	4.7613	4.4524	4.0723	3.5443	1.4774	0.0005	0.0005
		A	19.8004	18.4686	16.3394	13.3739	6.3584	0.0466	0.0466
	$E > 0.001$	R	14.3766	14.0458	13.5518	12.7288	9.8072	0.0188	0.0188
		G	42.5210	41.7433	40.6967	38.6374	31.4014	1.1439	1.1430
		B	5.7734	5.6461	5.4825	5.2503	4.7904	0.0421	0.0421
		A	27.5227	26.7311	25.5123	23.6230	19.5327	2.6603	2.6590
Boston Teapot	$E > 0.01$	R	0.5924	0.4724	0.3095	0.1196	0.0022	0.0033	0.0033
		G	11.8267	8.2691	3.0722	0.1066	0.0000	0.0015	0.0015
		B	25.9536	23.8144	17.4653	4.3946	0.0011	0.0115	0.0115
		A	23.3017	21.2913	18.7156	1.7756	0.0000	0.0200	0.0204
	$E > 0.005$	R	0.8045	0.7045	0.5579	0.3550	0.0596	0.0048	0.0048
		G	21.2495	15.9777	9.5949	2.3395	0.0000	0.0055	0.0055
		B	31.1572	31.4112	29.2872	22.1984	0.8952	0.0167	0.0159
		A	31.3087	29.5282	25.8252	22.4631	0.7086	0.0363	0.0359
	$E > 0.001$	R	1.1277	1.0888	1.0144	0.8604	0.5920	0.0192	0.0192
		G	34.1831	32.9758	30.1506	26.0377	7.0544	0.0241	0.0237
		B	36.2231	36.4501	35.9784	34.7233	32.0621	0.0592	0.0589
		A	36.2931	36.1502	35.8629	35.1987	32.4153	0.0874	0.0870
Engine	$E > 0.01$	R	27.6767	26.0282	21.8861	16.3762	4.6946	0.0908	0.0908
		G	26.9381	22.4487	17.1531	7.3850	0.0214	0.0823	0.0823
		B	26.4493	21.5112	16.6903	6.5777	0.0023	0.0739	0.0739
		A	4.9382	4.2154	3.3066	2.0843	0.1951	0.0197	0.0192
	$E > 0.005$	R	31.9896	31.2906	28.9227	24.2404	11.4493	0.0654	0.0654
		G	32.0939	29.6443	25.0810	15.2176	3.8856	0.0614	0.0614
		B	31.3909	28.6346	24.8194	14.2553	2.3538	0.0598	0.0598
		A	6.8686	6.3104	5.5831	4.3518	1.9039	0.0231	0.0231
	$E > 0.001$	R	35.7958	35.6311	35.1435	34.0790	28.5359	4.9337	4.9337
		G	35.6548	35.2111	34.1726	31.7427	23.1179	0.0885	0.0863
		B	35.4738	34.9264	34.0988	31.3148	21.6532	0.1003	0.0975
		A	10.2372	9.9023	9.4298	8.7138	7.0507	0.3259	0.3259

Tabela D.2 – Testes de performance em CPU (tempo em segundos para gerar um quadro).

	Blunt Fin	Bonsai	Boston Teapot	Engine
Riemann ($h = 0.5$)	6.53	16.23	17.97	8.86
Riemann ($h = 0.4$)	8.05	20.03	21.77	10.91
Riemann ($h = 0.3$)	10.52	26.31	27.87	14.33
Riemann ($h = 0.2$)	15.47	38.94	40.17	21.23
Riemann ($h = 0.1$)	30.33	76.80	76.55	41.70
Coupled ($\epsilon = 0.01$)	8.98	23.12	21.41	14.22
Decoupled ($\epsilon = 0.01$)	9.28	23.61	21.80	14.00
Coupled ($\epsilon = 0.005$)	9.36	25.70	21.97	15.30
Decoupled ($\epsilon = 0.005$)	9.67	26.30	22.28	15.02
Coupled ($\epsilon = 0.001$)	11.52	31.56	23.56	18.37
Decoupled ($\epsilon = 0.001$)	12.14	32.30	23.64	18.06

Tabela D.3 – Tabela de performance em GPU (quadros por segundo).

	Blunt Fin	Bonsai	Boston Teapot	Engine
Riemann ($h = 0.5$)	603	68	149	156
Riemann ($h = 0.4$)	494	55	120	126
Riemann ($h = 0.3$)	390	42	91	96
Riemann ($h = 0.2$)	270	28	62	65
Riemann ($h = 0.1$)	143	14	32	34
Coupled ($\epsilon = 0.01$)	375	18	42	42
Decoupled ($\epsilon = 0.01$)	344	17	38	40
Coupled ($\epsilon = 0.005$)	325	15	40	39
Decoupled ($\epsilon = 0.005$)	300	14	37	37
Coupled ($\epsilon = 0.001$)	201	12	34	30
Decoupled ($\epsilon = 0.001$)	185	11	32	29

Tabela D.4 – Erros RGBA para o modelo Bonsai com $h_{min} = 0.4$ e $h_{max} = 4.0$.

	$E > 0.001$			
	R	G	B	A
Riemann ($h = 0.1$)	9.8072	31.4014	4.7904	19.5327
Coupled	5.0038	10.5680	2.3044	3.9481
Decoupled	5.0038	10.5674	2.3044	3.9477

Tabela D.5 – Performance em CPU e GPU para o modelo Bonsai com $h_{min} = 0.4$ e $h_{max} = 4.0$.

	CPU (s)	GPU (fps)
Riemann ($h = 0.1$)	76.80	14
Coupled ($\epsilon = 0.001$)	16.20	35
Decoupled ($\epsilon = 0.001$)	16.52	34