UNIVERSIDADE FEDERAL DE ALFENAS
INSTITUTO DE CIÊNCIAS EXATAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

Luís Theodoro Oliveira Camargo

A UTILIZAÇÃO DA TECNOLOGIA CUDA EM TÉCNICAS DE RAY TRACING PARA RENDERIZAR IMAGENS TRIDIMENSIONAIS

Alfenas, 30 de junho de 2010.

UNIVERSIDADE FEDERAL DE ALFENAS
INSTITUTO DE CIÊNCIAS EXATAS
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

A UTILIZAÇÃO DA TECNOLOGIA CUDA EM TÉCNICAS DE RAY TRACING PARA RENDERIZAR IMAGENS TRIDIMENSIONAIS

Luís Theodoro Oliveira Camargo

Monografia apresentada ao Curso de Bacharelado em Ciência da Computação da Universidade Federal de Alfenas como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Paulo Alexandre Bressan

Alfenas, 30 de junho de 2010.

Luis Theodoro Oliveira Camargo

A UTILIZAÇÃO DA TECNOLOGIA CUDA EM TÉCNICAS DE RAY TRACING PARA RENDERIZAR IMAGENS TRIDIMENSIONAIS

A Banca examinadora abaixo-assinada aprova a monografia apresentada como parte dos requisitos para obtenção do título de Bacharel em Ciência da Computação pela Universidade Federal de Alfenas.

Prof. M.e Luiz Eduardo da Silva

Universidade Federal de Alfenas

Prof. M.e Tomás Dias Sant' Ana

Universidade Federal de Alfenas

Prof. Dr. Paulo Alexandre Bressan

Universidade Federal de Alfenas

Alfenas, 30 de junho de 2010.

À minha mãe, meu pai, meu irmão e a minha tia Deolinda.

AGRADECIMENTO

Agradeço primeiramente a minha mãe pela dedicação incansável e por todo o amor que recebo dela. Agradeço ao meu pai por ter me apoiado em minhas decisões e mostrar meus erros. Ao meu irmão, meu eterno amigo, que sempre estará ao meu lado. Agradeço a minha tia Deolinda que para mim sempre foi como uma segunda mãe. Ao meu avô por seu exemplo de seriedade. Agradeço a todos os meus familiares que de forma direta ou não contribuíram para eu estar aqui. Agradeço a minha namorada Ana Paula que compartilha comigo meus problemas e minhas alegrias. Não há palavras para descrever o carinho que sinto por todas as pessoas aqui referidas.

Agradeço a Prof.ª Melise pela oportunidade que tive de trabalhar com ela durante a Iniciação Científica. Agradeço ao Prof.º Bressan que mesmo sem me conhecer aceitou me orientar nesse projeto e me ajudou muito durante a pesquisa. Agradeço também a todos os professores que compartilharam comigo seus conhecimentos que irão me acompanharam por toda vida.

Agradeço aos meus amigos que sempre estiveram ao meu lado. Aos colegas da faculdade, sempre dispostos a ajudar. Aos meus três grandes amigos e companheiros de república Danilo, José Alexandre e Neubio que durante esses anos de faculdade foram minha família em Alfenas.

Por fim, sou grato a todos que de algum modo me ajudaram a alcançar meus objetivos.

"O futuro não pode ser previsto, mas pode ser inventado. É a nossa habilidade de inventar o futuro que nos dá esperança para fazer de nós o que somos."

Dennis Gabor

RESUMO

O vertiginoso crescimento da indústria de entretenimento digital impulsionou a criação de placas gráficas com alto poder computacional, grande quantidade de memória, muitos núcleos de processamento e gerenciamento de threads, conhecidas como GPUs (Graphics Processing Units) programáveis. Os pesquisadores tem percebido que essas características de hardware podem ser utilizadas em inúmeros outros problemas científico, como por exemplo, a visualização volumétrica de dados. Esta, por sua vez, consiste na técnica de gerar uma representação gráfica para conjuntos de dados complexos, o que auxilia especialistas das mais diversas áreas na análise dos dados.

As placas gráficas que possuem a tecnologia Compute Unified Device Architecture (CUDA) disponibilizam um conjunto de instruções que permitem ao programador desenvolver aplicações através da linguagem C que se beneficiam do processamento das GPUs. Essas placas possuem centenas de processadores, onde cada um pode realizar uma operação diferente sobre um conjunto de dados específicos.

Neste trabalho será explorada a utilização da técnica de Ray Tracing sobre a tecnologia CUDA para visualizar dados volumétricos de imagens médicas.

Palavras-Chave: Visualização volumétrica, Ray Tracing em imagens médicas, Placas gráficas programáveis.

ABSTRACT

The dizzying growth of the digital entertainment industry spurred the creation of graphics cards with high computational power, large amount of memory, many processing cores and management of threads, known as programmable GPUs.

Graphics cards that have the technology Compute Unified Device Architecture (CUDA) offers a set of instructions that allow the programmer to develop applications using C language to benefit from the processing of Graphics Processing Units (GPUs).These cards have hundreds of processors, where each one can perform a different operation on a set of specific data.

In this paper we explore the implementation of the technique of Ray-Tracing on CUDA technology to visualize volumetric data of medical images.

Keywords: Volume visualization, Ray Tracing in medical images, Programmable graphics cards.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIAÇÕES

API	*Application Programming Interface* (Interface de Programação de Aplicativos)
CPU	*Central Processing Unit* (Unidade Central de Processamento)
CUDA	*Compute Unified Device Architecture* (Arquitetura Unificada de Dispositivos Computacionais)
FPS	*Frames per Second* (Quadros por Segundo)
GPGPU	*General Purpose Graphics Processing Unit* (Unidade de Processamento Gráfico de Propósito Geral)
GPU	*Graphic Processing Unit* (Unidade de Processamento Gráfico)
MPI	*Message Passing Interface* (Interface de Passagem de Mensagem)
PC	*Personal Computer* (Computador Pessoal)
RAM	*Random Access Memory* (Memória de Acesso Randômico)
SIMT	*Single Instruction Multiple Thread* (Instrução Única Múltiplos *Thread)*

SUMÁRIO

1
Introdução

Este capítulo apresenta na seção 1.1 a justificativa e a motivação para a realização deste projeto. A seção 1.2 a problematização do projeto. Os objetivos gerais e específicos são apresentados na seção 1.3. A organização da monografia é apresentada na seção 1.4. Este capítulo tem como objetivo apresentar para o leitor a proposta do trabalho assim com o que será realizado.

A computação gráfica nos permite recriar ou visualizar ambientes que não seriam possíveis fisicamente. Esse poder de transformar o abstrato em imagem que a computação gráfica oferece pode ser utilizado em diversos setores da sociedade, como por exemplo, em uma tomografia computadorizada que resulta em centenas de imagens bidimensionais do interior de um paciente. A técnica de visualização volumétrica permite organizar essas imagens e gerar um modelo tridimensional do paciente no computador, podendo assim permitir ao médico analisar o conteúdo da tomografia de vários ângulos.

Essa técnica também pode ser utilizada na busca de bolsas de petróleo por sismologia, que consiste na emissão de ondas de choques que refletem nas rochas e são captadas por sensores, a análise dos dados obtidos podem indicar a presença de petróleo. A análise dessas informações pode ser facilitada utilizando a técnica de visualização volumétrica para reconstruir o ambiente permitindo ao geólogo visualizar o subsolo, ao invés de ter que analisar as informações referentes ao subsolo numericamente.

A técnica de visualização volumétrica está inserida no contexto de problemas computacionais que podem ter seu processamento paralelizável. Essa classe de problemas tem se beneficiado da evolução na área de sistemas distribuídos, do surgimento de processadores com mais de um núcleo e com o surgimento de placas gráficas programáveis, sendo este o objeto de estudo deste trabalho.

Anos atrás uma placa gráfica possuía uma unidade de processamento gráfico (GPU) com funções fixas, construído sobre uma pipeline gráfica que pouco implementava recursos tridimensionais. Atualmente, as GPUs tem evoluído com processadores programáveis e com interfaces de programação de aplicações (API) bem elaboradas que realmente focam na programabilidade[1]. Embora as GPUs sempre tenham sido construídas voltadas para o mercado de entretenimento, o desempenho alcançado nos últimos anos está atraindo pesquisadores das mais variadas áreas, pois permite que experimentos científicos sejam executados em computadores pessoais (PCs). Esses equipamentos possuem um custo menor e permitem um investimento inicial baixo com aumento gradual do poder computacional ao longo do projeto.

Exemplos de trabalhos que utilizam GPU programáveis podem ser encontrados na visualização de imagens tridimensionais, na simulação molecular, no processamento de imagens, na animação de partículas, além dos próprios algoritmos de renderização[2] geométrica da computação gráfica. Em Ronghua *et al* (2008), imagens resultantes de tomografia computadorizada formam volumes tridimensionais e são processadas por placas gráficas NVIDIA GeForce 6800 aumentando a taxa de atualização de quadros por segundo. Em Walters *et al* (2008), simulações de dinâmica molecular baseadas em Lennard-Jones são aceleradas 100 vezes por GPUs utilizando uma técnica de divisão do problema discutida no artigo. A visualização de órbitas moleculares em GPU alcança um desempenho 125 vezes mais rápido que o processamento em CPU, como mostrado em Stone *et al* (2009). Uma demonstração de melhora de desempenho também é apresentada em Popov *et al* (2007) com a renderização de vários modelos geométricos através de técnicas de Ray Tracing, conseguindo chegar ao processamento de 16 milhões de raios por segundo para cenas consideravelmente complexas.

Especificamente, placas gráficas que possuem a tecnologia CUDA disponibilizam ferramentas de desenvolvimento em C/C++, bibliotecas de funções

xxvixxvi———————————

[1] *Programabilidade* – capacidade de implementação de programas

Renderizar – é uma palavra proveniente do verbo "*to render*" adaptada para o português que significa transformar um conjunto de dados n – dimensionais em uma imagem bidimensional por meio de técnicas de projeção

[2] *Renderização* – ato ou ação de renderizar

e um mecanismo de abstração de hardware que esconde particularidades do hardware da GPU do desenvolvedor (Halfhill, 2008). Apesar da tecnologia CUDA exigir que o desenvolvedor escreva um código específico para processamento paralelo, ela não requer o gerenciamento de threads o que fortemente simplifica o modelo de programação.

Além disso, o código escrito é portável para outras GPUs da série, o que não acontece com outras famílias de GPUs. Essas placas possuem centenas de processadores em que cada um realiza operações sobre um conjunto de dados específicos e seus resultados podem ser sincronizados em memórias internas.

1.1 Justificativa e Motivação

Placas gráficas programáveis permitem utilizar seus recursos disponíveis, que seriam utilizados normalmente apenas para o processamento gráfico, para resolver problemas de outra natureza. Essa abordagem se mostra muito eficiente quando aplicada a problemas de natureza paralelizável, pois as placas gráficas possuem centenas de unidades de processamentos que realizam operações independentes sobre conjuntos de dados diferentes.

A utilização desta tecnologia implica em adaptar o código utilizado pela CPU para que seja utilizado na GPU. A programação com placas que possuem a tecnologia CUDA envolve o conhecimento de recursos disponíveis que devem ser levados em consideração para realizar a adaptação e permitir a utilização completa dos recursos disponibilizados pela GPU.

Embora as utilizações das placas gráficas programáveis tenham um bom desempenho computacional, essa abordagem não pode ser utilizada em todas as classes de problemas. Como as bases de dados volumétricos de imagens médicas possuem forte coerência espacial e funcional justifica-se sua utilização, pois esses dados podem ser separados para cada linha de execução paralela de maneira a melhorar o desempenho. A utilização da tecnologia CUDA permitirá a construção de uma aplicação paralelizável em computadores convencionais, ou seja, computadores de fácil aquisição.

1.2 Problematização

A visualização volumétrica é definida como sendo o processo de representação de um conjunto de dados em forma de imagens tridimensionais. Esse processo tem um alto custo computacional e é altamente paralelizável. Essas duas características do problema, aliado ao advento das placas gráficas programáveis, levam a seguinte questão:

Como utilizar os recursos de placas gráficas programáveis nas técnicas de Ray Tracing para visualizar imagens médicas?

1.3 Objetivos

1.3.1 Gerais

Desenvolver um aplicativo para visualizar tridimensionalmente imagens médicas de forma interativa com a utilização da tecnologia CUDA

1.3.2 Específicos

- Estudar fundamentos de visualização científica.
- Estudar a tecnologia CUDA
- Estudar a técnica de Ray Tracing em imagens médicas
- Criar um aplicativo de visualização volumétrica de imagens médicas que se utilize da tecnologia CUDA
- Avaliar qual a melhor forma de utilizar os recursos disponíveis

1.4 Organização da Monografia

No Capítulo 1 foi proporcionado uma visão geral sobre o projeto assim como os objetivos definidos para este. O Capítulo 2 irá apresentar a revisão bibliográfica do projeto, que inclui um estudo sobre os temas que serão abordados neste trabalho e uma revisão sobre outros trabalhos que possuem o mesmo tema. A descrição do Projeto apresentado neste trabalho será descrito no Capítulo 3, onde também serão apresentados o experimento realizado. A conclusão do projeto é apresentado no Capítulo 4.

2
Revisão Bibliográfica

Este capítulo apresenta na Seção 2.1 um estudo sobre a visualização volumétrica. Na Seção 2.2 é descrito a tecnologia CUDA. A visualização volumétrica é uma área da computação gráfica que visa tornar mais prático a análise de dados complexos através da geração de imagens que represente esses dados, este capítulo proporciona uma visão geral da visualização volumétrica assim como as fases do processo, os algoritmos utilizados e a utilização de algoritmos paralelos para sua execução. Também é apresentado neste capítulo a tecnologia CUDA implementada nas placas gráficas que permite uma programação utilizando tais placas de uma maneira mais natural e familiar ao programador.

2.1 Visualização Volumétrica

Visualização é uma ferramenta para a interpretação através da geração de imagens originadas de informações de conjuntos de dados complexos e multidimensionais. Denomina-se visualização científica quando estes conjuntos de dados representam fenômenos complexos e o objetivo é a extração de informações científicas relevantes. (Mccormick,1987).

Em termos gerais a visualização volumétrica pode ser descrita como o processo no qual a descrição abstrata de uma cena é convertida em imagem, ou seja, os dados são transformados em imagens.

Os dados quando são relacionados a uma região de objeto podem ser definidos como dados volumétricos. Assim a visualização volumétrica pode ser conceituada como a classe de métodos de visualização relacionada com a representação, manipulação e visualização de conjuntos de dados volumétricos (Paiva, 1999).

Os conjuntos dos dados que formam o volume são um conjunto S de amostras (x,y,z,v), sendo v o valor de alguma propriedade mensurável dos dados

ou atributo, como densidade, e (x,y,z) a sua localização no espaço tridimensional (Kaufman 1994).

Na maior parte das vezes esses dados são armazenados na forma de grades tridimensional onde um ou mais valores escalares, ou vetoriais, podem ser armazenado em cada ponto da grade, cada elemento da grade é definido como um vóxel (*volume element*).

Vóxel é a unidade mínima de um dado volumétrico. Pode-se considerar o vóxel como uma amostra no espaço tridimensional de um determinado volume de dados. Desta forma o vóxel é caracterizado por uma posição espacial e corresponde a uma ou mais grandezas numéricas associadas a ele que representem propriedades próprias do volume apresentado.

Outro conceito muito utilizado na visualização volumétrica é célula vóxel que representa um conjunto de vóxeis adjacentes conectados segundo uma topologia arbitrária definindo um espaço fechado.

A forma mais comum de célula vóxel é um cubo, em que cada um dos vértices é definido por um vóxel, outras abordagens são utilizadas na literatura. A Figura 1 ilustra esses conceitos apresentados.

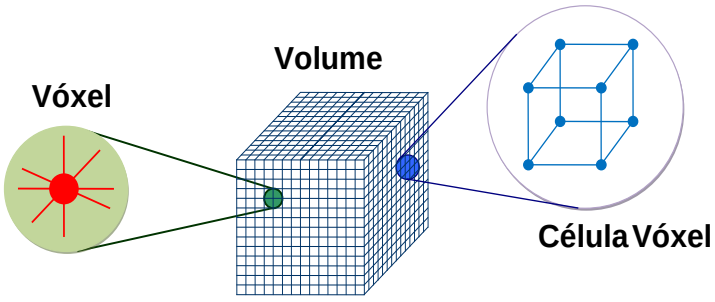

Figura 1 Vóxel, volume e célula vóxel (Zuffo, 1997).

Os dados volumétricos podem ser classificados de acordo com a sua origem:

- **Simulação**: dados obtidos a partir de uma modelagem matemática;
- **Aquisição**: dados resultantes da observação de um objeto da natureza e convertido para o conjunto de dados.

As duas principais técnicas usadas em visualização volumétrica são renderização volumétrica por isosuperfície e direta. A diferença entre estas técnicas consiste no tipo do elemento central utilizado na renderização. Enquanto

renderização por isosuperfície desenha superfícies geométricas, a renderização direta desenha o resultado de uma função aplicada sobre a base de dados. A Figura 2 apresenta a taxonomia de Visualização Volumétrica de Kaufman (1990).

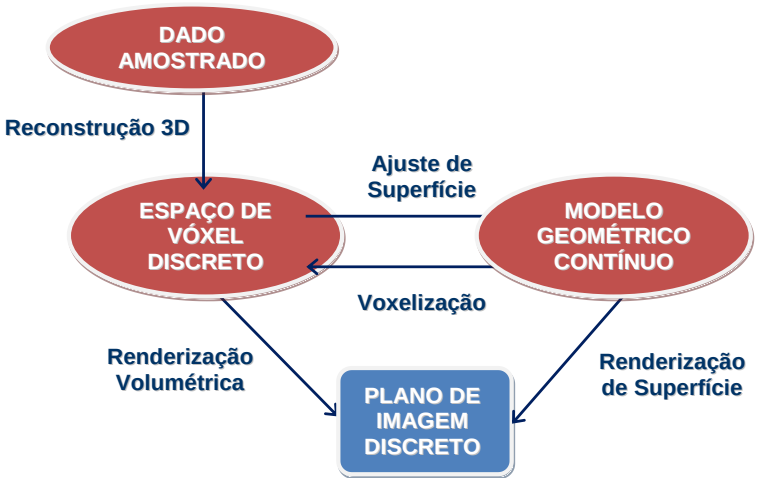

Figura 2 Taxonomia de visualização volumétrica (Kaufman, 1990).

2.1.1 Pipeline da Visualização

Em Carneiro (2000) o processo de visualização volumétrica é decomposto em quatro etapas (Figura 3) o que nos permite entender melhor como realizar a visualização e melhorar o processo. No entanto apenas as três últimas etapas são implementadas, pois consideremos apenas a visualização dos dados pré processados, assim não sendo necessário implementar a primeira fase que é a aquisição dos dados.

Aquisição: É o processo de obtenção dos dados, que geralmente é realizado por meio de equipamentos sensoriais apropriado a exemplo de tomógrafos, Raio–X, sismógrafos, etc. Além do processo de obter os dados, necessitamos processar as fatias obtidas de modo a melhorar as características como contraste e definir faixa de valores.

Classificação: Esse processo é um dos mais complicados, pois na maioria das vezes é feita por pessoas que não tem muitos conhecimentos do processo de visualização, mas que necessariamente precisam conhecer muito bem o objeto em estudo. Esse processo tem por objetivo identificar os componentes internos ao

volume. Por exemplo, caso os dados sejam obtidos a partir de uma tomografia computadorizada um médico poderá identificar qual componente representa como músculo, osso ou pele. Quando são utilizados algoritmos de renderização direta a classificação envolve em definir a relação dos valores do volume com as cores que serão apresentadas ao usuário. No caso de renderização por isosuperficie é necessário definir quais os valores de limiariazação serão poligonizados e mostrados.

Iluminação: Para criar efeito de profundidade e realçar as características do volume pode ser utilizado um modelo de iluminação para o objeto. O processo de iluminação pode proporcionar mais realismo à cena, mas necessitam de mais processamento. No caso de imagens médicas esse grau de realismo não é necessário por isso são utilizados métodos mais simples de iluminação.

Projeção: Essa etapa compreende a projeção dos vóxels na janela de visualização, assim compondo a imagem a ser visualizada, nessa etapa é onde ocorre a remoção de áreas escondidas e permite ao usuário selecionar o formato da visualização mais apropriado definindo o corte que deseja visualizar ou rotacionar o objeto.

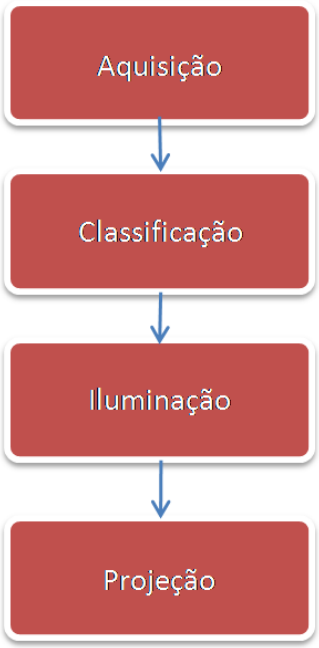

Figura 3 Etapas do processo de visualização

2.1.2 Renderização Volumétrica Direta

Nesta seção serão apresentados os algoritmos de renderização direta, esses algoritmos tem como característica não utilizar nenhuma representação intermediária para construir a imagem, mas tem um alto custo computacional.

A técnica renderização direta utiliza raios que são traçados entre o espaço do objeto e o plano do observador. Um raio é traçado para cada pixel da tela, onde será apresentada a imagem resultante, na direção e sentido do observador. O volume é então amostrado em intervalos regulares ao longo dos raios, onde os valores das amostras são compostos até que uma determinada função seja satisfeita ou que os raios saiam do espaço do objeto. A cor resultante da função ao longo do raio lançado é a cor final do pixel correspondente na tela.

Os métodos de renderização volumétrica direta podem ser classificados em duas categorias (Drebin; Carpenter & Hanrahan, 1988): objeto-imagem (*forward*) e imagem-objeto (*backward*). A composição objeto-imagem (*forward*) projeta vóxeis individuais no plano da imagem. A composição imagem-objeto (*backward*) lança raios do plano da imagem até o dado volumétrico. Composição imagem-objeto (*backward*) permite interromper antecipadamente o procedimento se um limiar foi alcançado. Já a composição objeto-imagem (*forward*) pode simplificar o processo, mas não permite interromper o processo com antecedência. Em ambos, o fundo sempre tem opacidade de valor unitário.

A visualização volumétrica faz uso extensivo de técnicas de interpolação numérica para que mesmo os pontos onde o raio não atravesse um vóxel, mas uma célula vóxel possa ser representada na imagem. Normalmente a função de interpolação é tridimensional, mas devido ao custo computacional, os algoritmos de renderização também podem usar interpolação bidimensional. Serão apresentados dois algoritmos de renderização direta o algoritmo de Ray Tracing e o de Shear-Warp.

2.1.2.1 Ray Tracing

O algoritmo de Ray Tracing percorre um caminho orientado pelos píxeis da imagem até a base de dados. Para cada píxel, um raio é traçado em direção aos dados onde pontos amostrados em sua extensão determinam cor e opacidade.

A função aplicada sobre o ponto que o raio encontra o volume pode possuir diferentes abordagens e depende das características da aplicação. Algumas implementações conhecidas podem ser citadas: primeiro valor, média das amostras, máxima intensidade e acumulativa. Estas implementações pode ser facilmente entendidas considerando os valores dos vóxeis amostrados ao longo do raio.

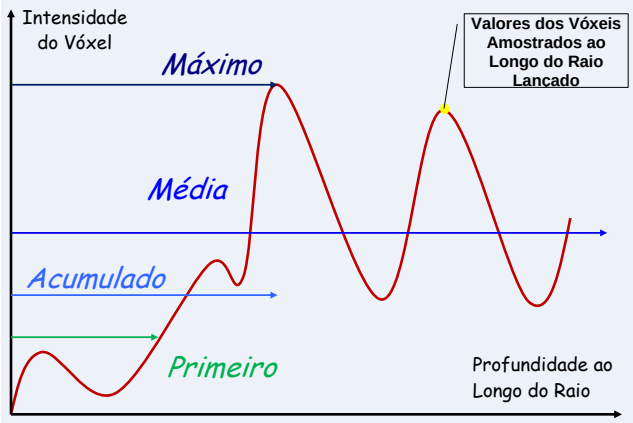

Figura 4 Funções do Ray Tracing

A Figura 4 considera este caminho e apresenta os valores dos píxeis para cada implementação. A função de primeiro valor considera o valor do primeiro vóxel encontrado numa dada faixa de valores, assumindo este como sendo o valor do píxel. A média das amostras é calculada pelo caminho completo do raio. A máxima intensidade corresponde ao maior valor encontrado no caminho percorrido pelo raio. E finalmente, a implementação acumulativa considera a opacidade atribuída a cada vóxel e termina quando a cor do pixel torna-se totalmente opaca (opacidade igual a um) ou o raio atravessa toda a base de dados.

A implementação acumulativa do Ray Tracing é a mais importante delas, pois expressa melhor as características internas da base de dados. Além disso, valores de opacidades podem ser atribuídos interativamente na fase de classificação dos vóxeis para melhorar a visualização do objeto de estudo. Nesta implementação, o término antecipado do cálculo do raio é um poderoso método de aceleração do Ray Tracing, o qual consiste em finalizar a cor do pixel quando a opacidade acumulada alcançou valor unitário.

2.1.2.2 Shear-Warp

O algoritmo de Shear-Warp parte da ideia de transformar o volume de dados de modo a simplificar a etapa de projeção do processo de visualização. Essa simplificação ocorre através do cisalhamento das fatias do volume, de modo que os raios de visão sejam perpendiculares as fatias do volume. Esse processo facilita a projeção das fatias, pois essas são acessadas na ordem em que foram armazenadas. Isso não ocorre no Ray Tracing. O cisalhamento apenas translada as fatias do volume, computacionalmente isso tem um custo muito baixo. Esse processo de cisalhamento cria uma imagem intermediária distorcida, que precisa ser corrigida para a visualização do usuário (transformação de *warp*) Figura 5.

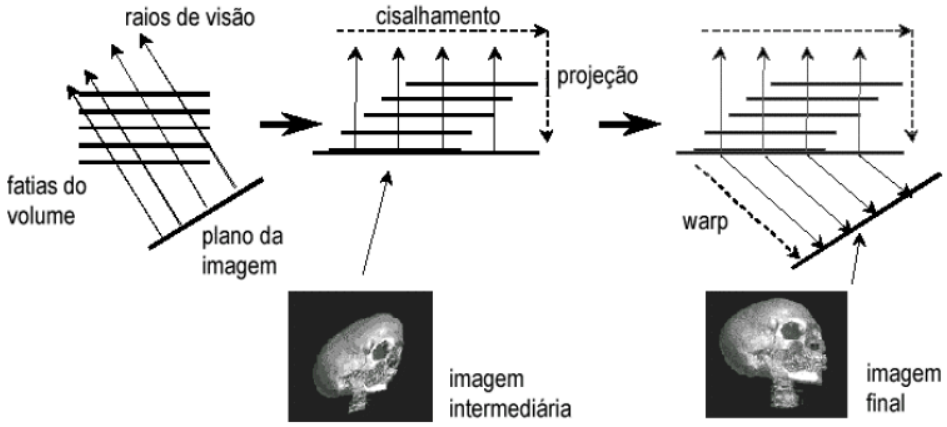

Figura 5 Shear-Warp (Carneiro 2000)

A vantagem principal deste processo é que a dimensão da imagem intermediária é igual ao tamanho do volume e do fator cisalhamento, independente da resolução da imagem final. As dimensões do volume normalmente são menores que a resolução da imagem final, o algoritmo de Shear-Warp leva uma grande vantagem em relação ao Ray Tracing em relação ao desempenho.

2.1.3 Visualização Volumétrica Paralela

A computação paralela tem crescido devido ao surgimento de processadores com mais de um núcleo e ao aumento da velocidade de transmissão de dados. E com o advento das placas gráficas programáveis (GPGPU – *General*

Purpose Graphic Processing Unit) surgiram mais plataformas que possibilitam a programação paralela e possibilitam o ganho de desempenho.

O algoritmo de Ray Tracing possui fortes traços de algoritmo paralelo, pois podemos atribuir cada raio traçado para uma linha de processamento paralela.

A tarefa de se criar tal algoritmo de renderização paralela eficiente não é trivial devido aos problemas que são introduzidos pela programação paralela como: a comunicação entre tarefas, o atraso na execução das tarefas, computação redundante e a replicação de dados (Bressan, 2004).

Na literatura existem diversos trabalhos que tratam sobre a renderização paralela e utilizam de diversas abordagens.

Em Farias e Bentes (2004) eles descrevem um algoritmo paralelo que executa sobre clusters de computadores para dados irregulares, eles dividem a imagem em retângulos de tamanho fixo denominados tiles, cada um dos tiles recebe um identificador e através desse identificador é possível determinar qual porção deve ser computada. A comunicação entre os nós do cluster [1] é feita através da biblioteca de troca de mensagens MPI (*Message Passing Interface*).

Em Bressan (2004) é apresentado a proposta de uma aplicação de visualização volumétrica que utiliza cluster de computadores, utilizando também a biblioteca MPI. No trabalho os dados assim como em Farias & Bentes (2004) também são particionados para serem divididos entre os clusters, mas a diferença é a característica dos dados utilizados em Bressan (2004), ele utiliza dados médicos que possuem alta coerência espacial e funcional enquanto no trabalho de Farias e Bentes (2004) os dados são irregulares.

Ambas as propostas anteriores utilizam a abordagem de clusters computacionais com a troca de comunicação entre os computadores. Abordagem que se utiliza da arquitetura de processadores paralelos pode ser visto em Kim & Jaja (2009) que apresentam uma aplicação de visualização volumétrica aplicada sobre a arquitetura dos processadores *Cell* que constituem de 9 núcleos heterogêneos, onde cada núcleo pode ficar responsável por tarefas especificas de maneira paralela.

xxxviiixxxviii

[1] *Cluster* – Neste trabalho esta palavra significa aglomerados de computadores.

Outra abordagem muito utilizada é utilizar GPUs para realizar o processamento, isso pode ser feito utilizando-se de primitivas da API OpenGL, o que implica em transformar todos as operações do algoritmo em primitivas gráficas. Outra possibilidade é através da utilização de GPGPU, placas gráficas programáveis que permitem ao programador escrever códigos que serão executados pela GPU.

2.2 CUDA

O crescimento da indústria do entretenimento impulsionou a demanda por placas gráficas capazes de reproduzir imagens com grande quantidade de detalhes e alto grau de realismo, o processo para gerar imagens com essas qualidades necessita um alto poder computacional.

Com o objetivo de disponibilizar o poder computacional exigido pela indústria as placas gráficas necessitam realizar uma grande quantidade de cálculos, grande parte desses cálculos são operações de ponto flutuante o que fez que os projetistas desenvolvem-se GPUs que possam realizar um grande número dessas operações. Essa característica da GPUs permite que elas possuam desempenho melhor que as CPUs quando comparadas em relação à quantidade de operações de números de ponto flutuante realizados por segundo.

A diferença da evolução do poder de processamento das CPUs para as GPUs pode ser visto na Figura 6.

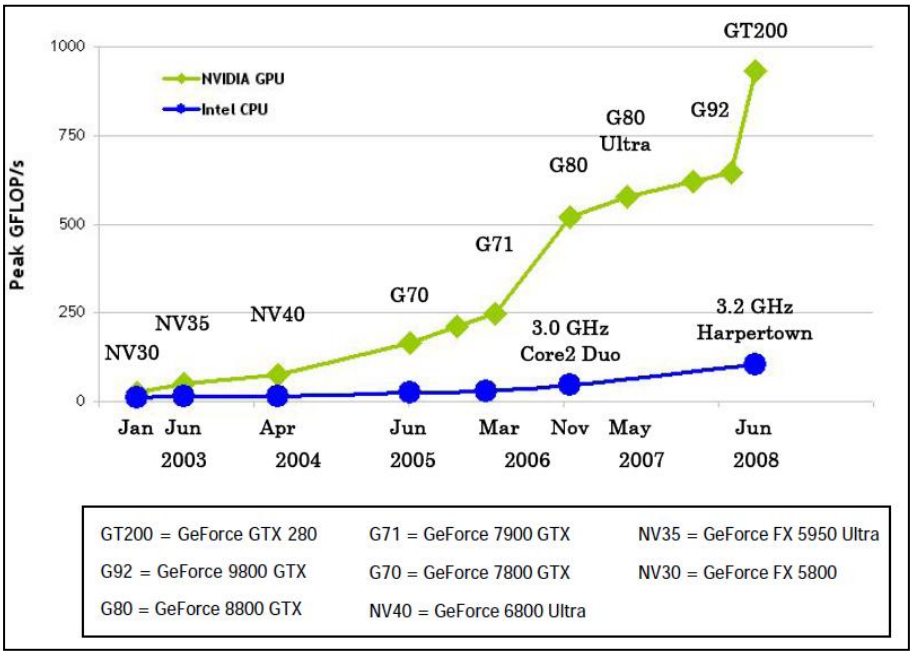

Figura 6 Evolução do poder de processamento das placas gráficas (NVIDIA, 2008)

Essa diferença do poder de processamento das GPUs em relação às CPUs existe devido à arquitetura de cada uma. Enquanto as CPUs necessitam trabalhar com um conjunto de instruções maior e mais generalista, as GPUs possuem um número maior de unidades de operações aritméticas, pois possuem um conjunto de instrução mais restrito e não necessitam de unidades controladoras tão sofisticadas quanto as CPUs a Figura 7 ilustra a arquitetura das CPUs e GPUs.

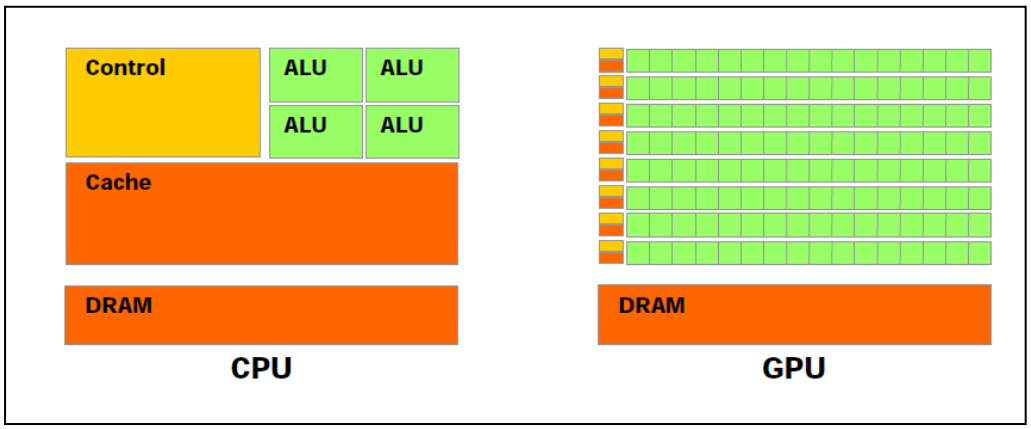

Figura 7 Arquitetura da GPU e CPU (Nvidia, 2008)

Pesquisadores perceberam que podem utilizar a capacidade de processamento das placas gráficas em outros problemas, mas para que fosse

possível fazer isso eles deveriam adaptar seus problemas transformando-os em problemas de computação gráfica e programando utilizando-se de APIs como a OpenGL, essa adaptação por muitas vezes não era trivial de realizar. Visando a possibilidade de aumentar a demanda por suas placas gráficas a NVIDIA desenvolveu a tecnologia CUDA de forma a permitir uma programação mais próxima a aquela que o programador está acostumado para CPU.

O CUDA é um modelo de programação paralelo e pode ser considerada uma extensão da linguagem C, ela permite ao programador definir funções que serão executadas paralelamente dentro da GPU. A vantagem do CUDA é abstrair do programador os detalhes do hardware onde o programa será executado e facilitar a programação paralela. O processo de abstração utilizado pelo CUDA é o mesmo que é utilizado nos processadores, onde o programador só necessita saber quais são as operações suportadas pelo hardware e oculta do programador detalhes de como as operações são implementadas e que permite que mesmo códigos que foram escritos antes da criação dos processadores utilizados atualmente possam ser executados sem muitos problemas, essa abstração facilita o processo de programação, pois tira do programador a necessidade de conhecer a estrutura do hardware de onde a aplicação será executada. Essa abstração também permite que a NVIDIA futuramente possa alterar seu hardware ou a maneira como as instruções são executadas sem que os códigos para a plataforma CUDA precisem ser reescrito para ser executado no novo hardware.

A tecnologia CUDA permite que o programador defina funções que serão executadas pelo GPU, essas funções são nomeadas *Kernel*. Quando uma função do tipo *Kernel* é chamada ela é executada N vezes paralelamente em N *threads*, ao invés de ser executada sequencialmente como aconteceria normalmente com a programação para CPU. A Figura 8 ilustra graficamente como seria a execução de um código heterogêneo escrito em CUDA que teria funções do tipo *Kernel* e funções que seriam executadas pela CPU.

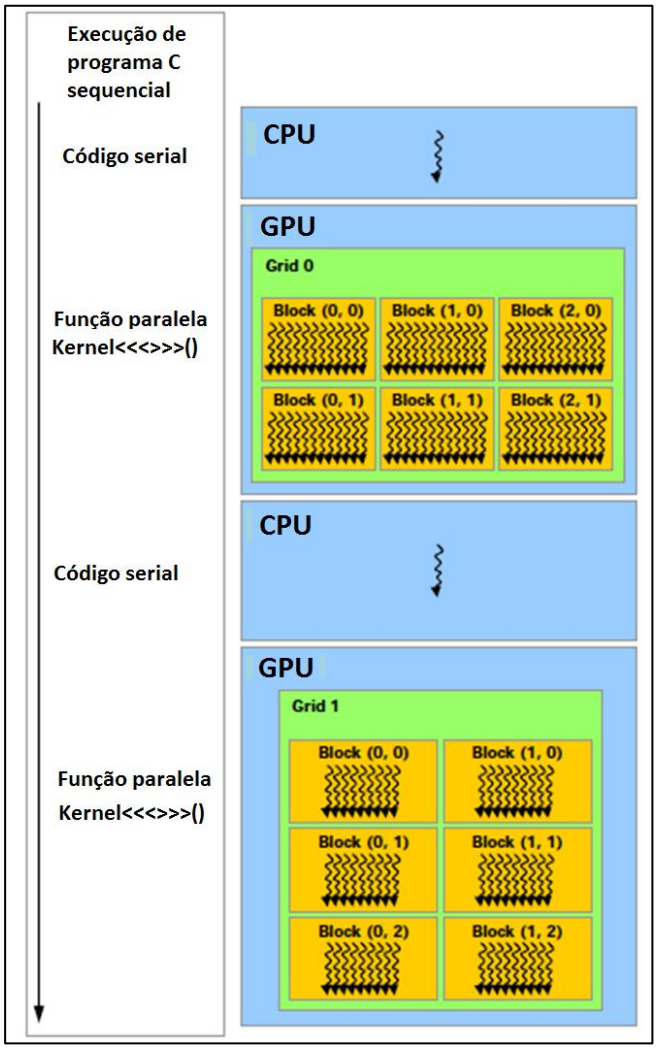

Figura 8 Execução de um código utilizando a tecnologia CUDA (NVIDIA, 2008)

No CUDA cada linha de execução é definida como sendo uma *thread* e é utilizada a arquitetura SIMT onde as *threads* podem ser organizadas em conjuntos de até 32 *threads* gerenciado por um multiprocessador.

A definição de uma função de *kernel* no CUDA é feita usando a declaração __globlal__ se especificando o numero de threads para cada chamada que devem ser definidos entre <<< >>> a Figura 9 ilustra um trecho de código escrito para CUDA.

```
__global__ void vecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    // Kernel invocation
    vecAdd<<<1, N>>>(A, B, C);
}
```

Figura 9 Exemplo de código CUDA (NVIDIA, 2008)

Cada *thread* recebe um identificador único que pode ser utilizado para realizar a operação, na Figura 9 é exemplificado a soma de dois vetores e o identificador da *thread* é utilizado como índice do vetor e o número de *threads* deve ser igual ao tamanho dos vetores.

3
Projeto

Este capítulo apresenta, na Seção 3.2 a base de dados que é utilizada no projeto. A Seção 3.3 apresenta a aplicação resultado do estudo. O resultado do experimento executado se encontra na Seção 3.4. Neste capítulo é descrito o projeto executado, desde a seleção dos dados utilizados, até a execução de experimentos para medir desempenho, passando pela implementação de funções que facilitem o processo de visualização e a criação de uma interface para o usuário.

3.1 Considerações Iniciais

Como foi descrito no Capítulo 1 na seção de Problematização o objetivo deste projeto é criar um aplicativo de visualização volumétrica utilizando-se a tecnologia de CUDA.

Para isso será utilizado como base um código exemplo disponível juntamente com o pacote de desenvolvimento CUDA disponibilizado pela NVIDIA. Apesar de esse código realizar o processo de Ray Tracing é necessário adaptá-lo para utilizar os dados médicos do projeto, além de ter que ser otimizado para o hardware utilizado. Como um dos objetivos desse projeto é a melhor utilização dos recursos disponíveis é importante a descrição do hardware utilizado. Este projeto foi desenvolvido utilizando-se um computador com processador Pentium® Core 2 Quad 2.83GHz, 8 GByte de memória RAM e placa gráfica NVIDIA GeForce GTX 285 com um 1GByte de memória.

3.2 Base de Dados

O volume de dados utilizados neste projeto é proveniente do projeto *Visible Human* (National Library of Medicine, 2010). O objetivo do projeto *Visible Human* é criar um banco de dados digital de imagens do corpo humano para a utilização em pesquisas médicas. O projeto foi definido em 1989 a fim de construir uma biblioteca digital de imagens volumétricas do corpo masculino e feminino. Os dados são disponibilizados para a utilização como referência em estudos e para criação de aplicativos.

As imagens utilizadas neste projeto são referentes ao corpo masculino, os dados foram obtidos em 1994 a partir da secção do corpo criogenizado, a Figura 10 é um exemplo de imagem obtida a partir de uma das secções da cabeça. A distância entre cada uma das imagens obtidas é de 1 milímetro, essa distância foi definida para que as imagens coincidam com as imagens obtidas através de tomografia computadorizada, as imagens obtidas possuem resolução de 2048 pixels por 1216 pixels onde cada píxel representa 0,33mm em tamanho. A cor é definida por 24 bits, um byte para cada componente vermelho, verde e azul do padrão RGB.

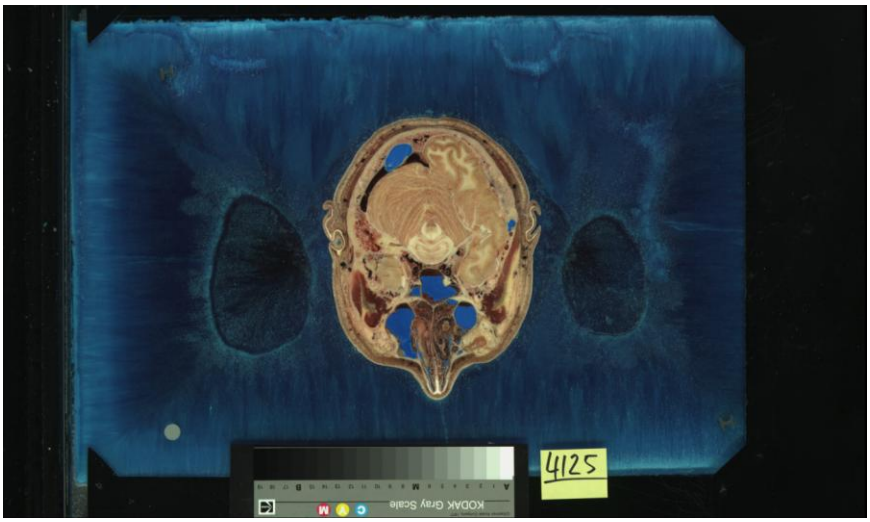

Figura 10 Exemplo de imagem do projeto Visible Human (National Library of Medicine, 2010)

Para que não houvesse perda de dados no armazenamento das imagens, estas foram armazenas em arquivos do tipo RAW. Esse tipo de arquivo contém

apenas o dado bruto, sem nenhum tipo de compactação, representam a imagem digital de maneira fiel, por isso são comumente chamadas de "negativos digitais" em referência ao tipo de armazenamento utilizado pelas antigas câmeras fotográficas analógicas. A Figura 11 ilustra um trecho dos dados contidos em um arquivo RAW.

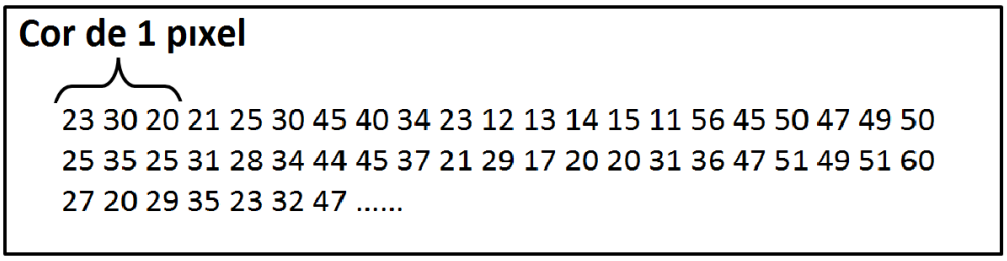

Figura 11 Exemplo de arquivo RAW

Este tipo de arquivo não possui em seu corpo cabeçalho com informações sobre a imagem, como normalmente ocorre com imagens digitais, nem mesmo informações sobre tamanho ou o padrão de cores utilizados. Essas informações deve ser especificas em outro arquivo, no caso dos dados do projeto *Visible Human* essas informações se encontram, em um arquivo texto que as acompanha, neste mesmo arquivo também é descrito o padrão de cores utilizadas, que neste caso é utilizado o padrão RGB.

Para este projeto foi definido como objeto de estudo apenas a cabeça, a seleção de apenas uma parte do corpo humano tenta representar uma possível real aplicação para o projeto em exames tomográficos, além da decisão pela utilização de um conjunto reduzido de dados se fez necessário devido a limitações da placa gráfica utilizada que serão discutidos mais a frente.

3.3 Aplicação

Uma das dificuldades encontradas em se realizar aplicações paralelas é definir qual será a parte do código executado em paralelo e como isso pode ser feito, neste projeto o processamento paralelo ocorre sobre a operação de Ray Tracing. Isso porque como foi descrito na seção 2.1.2.1 que cada raio lançado pode

ser atribuído a uma linha de processamento independente. A Figura 12 mostra como ocorre o fluxo do processamento da aplicação dividido entre a CPU e a GPU.

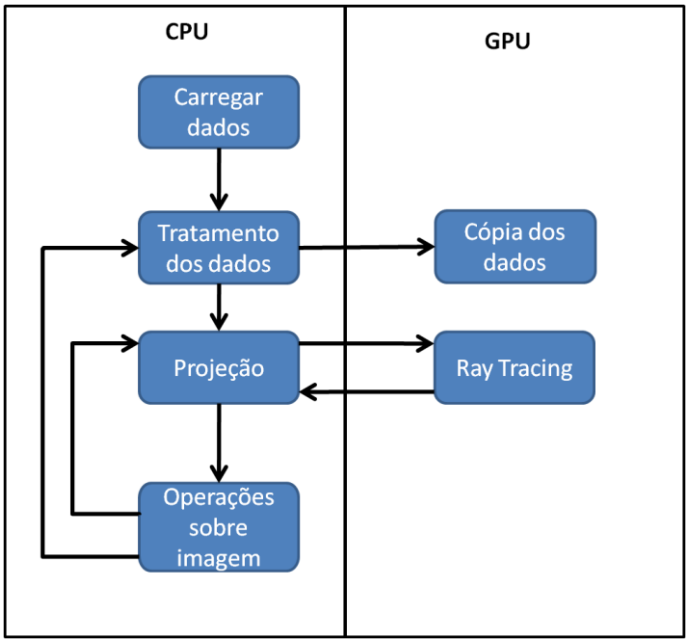

Figura 12 Fluxo de processamento

Nas próximas seções serão apresentadas de forma clara o que ocorre em cada fase de execução definida para a aplicação e as dificuldades encontradas na implementação de cada uma delas.

3.3.1 Carregar Dados

Como foi descrito anteriormente a base de dados utilizados neste projeto é referente cabeça humana, essa base de dados é composta por 250 imagens onde cada imagem é formada por 2048x1216 pixels e onde cada pixel é representado por três bytes (cores RGB). A partir dessas informações é possível determinar que cada imagem possui por volta de 7MB e que toda a base possui 1,8GB, esta informação sobre a base de dados se tornam importantes devido a necessidade de se alocar memória suficiente na placa gráfica para armazenar todos os dados que serão utilizados e a capacidade da memória da placa gráfica disponível para a execução do projeto ser de apenas 1GB.

A limitação da quantidade de memória implicou na decisão de utilizar apenas metade dos dados referente a cabeça humana neste projeto, essa decisão

não implica em perda da qualidade da imagem resultante devido a distância entre as imagens obtidas serem de apenas 1mm. No entanto, isto nos permite concluir que uma aplicação comercial que utilize a tecnologia CUDA deve levar em consideração o hardware utilizado e o volume de dados manipulado.

As imagens formam uma base de dados tridimensional, mas são organizadas na memória na forma de um vetor unidimensional onde cada três componentes do vetor representam a cor de um píxel. Essa mesma abordagem de armazenar os dados de forma linear é adotado também nas placas gráficas, por isso faz-se necessário a definição de uma estrutura que permita a placa interpretar o vetor como sendo um objeto tridimensional. O próprio conjunto de instruções CUDA contempla uma estrutura formada por três inteiros onde são armazenados informações sobre a altura, largura e a profundidade do volume, denominada `cudaExtent`. Apesar dessa fase já ser possível identificar a limitação da placa gráfica, neste momento só é realizada a leitura dos dados que são armazenadas na memória principal.

3.3.2 Tratamento dos Dados

Nesta fase ocorre a manipulação dos dados do volume, com todos os dados na memória da CPU e com as informações sobre o tamanho do mesmo. Podemos realizar operações sobre os dados de forma a melhorar a visualização do objeto de estudo, esse processo de melhoria da visualização será explicado quando for descrito a funcionalidade de corte na subseção de tratamento de eventos. Além disso, nesta etapa também ocorre o processo de alocação de memória na placa gráfica e a cópia dos dados para a placa. Para realizar a operação de alocação de memória que será utiliza pelo processamento na GPU é utilizado a estrutura `cudaExtent` que foi inicializada na fase anterior e nomeada como volumeSize. A Figura 13 ilustra o trecho do código que realiza a alocação de memória na placa e a transferência dos dados para a mesma.

```
cudaMalloc3DArray(&d_volumeArray, &channelDesc, volumeSize);
// copy data to 3D array
    cudaMemcpy3DParms copyParams = {0};
    copyParams.srcPtr   = make_cudaPitchedPtr((void*)h_volume, volumeSize.width*sizeof(uchar),
                                                  volumeSize.width, volumeSize.height);
    copyParams.dstArray = d_volumeArray;
    copyParams.extent   = volumeSize;
    copyParams.kind     = cudaMemcpyHostToDevice;
cudaMemcpy3D(&copyParams);
```

Figura 13 Trecho de código da responsável pela alocação de memória e cópia dos dados

3.3.3 Projeção

Cada um dos raios lançados pelo Ray Tracing "caminha" por todo o volume de dados e a cada passo do raio ele realiza uma amostragem do valor da célula vóxel que encontrou. Esses valores são acumulados durante todo o caminho do raio, o resultado desse valor acumulado é o valor que será atribuído ao pixel que será projetado na tela para o usuário. A distância entra as amostras feitas pelo raio influencia na qualidade da imagem resultante, quanto menor a distância entre as amostras maior será o grau de detalhismo da imagem. Em contrapartida será necessário uma quantidade maior de processamento que pode levar a uma queda na taxa de atualização da imagem que é apresentada para o usuário. Neste trabalho quando não for especificado o valor da distância entre os passos esse será de 0,001.

O valor definido para cada píxel é armazenado em uma matriz, que é utilizado pela função da biblioteca OpenGL que desenha cada um dos píxeis na tela. A partir da projeção do volume podemos agora tratar a imagem resultante de forma a tentar facilitar a visualização do objeto em estudo. A Figura 14 mostra o resultado da execução da aplicação sem a realização de nenhuma operação para tratar a imagem.

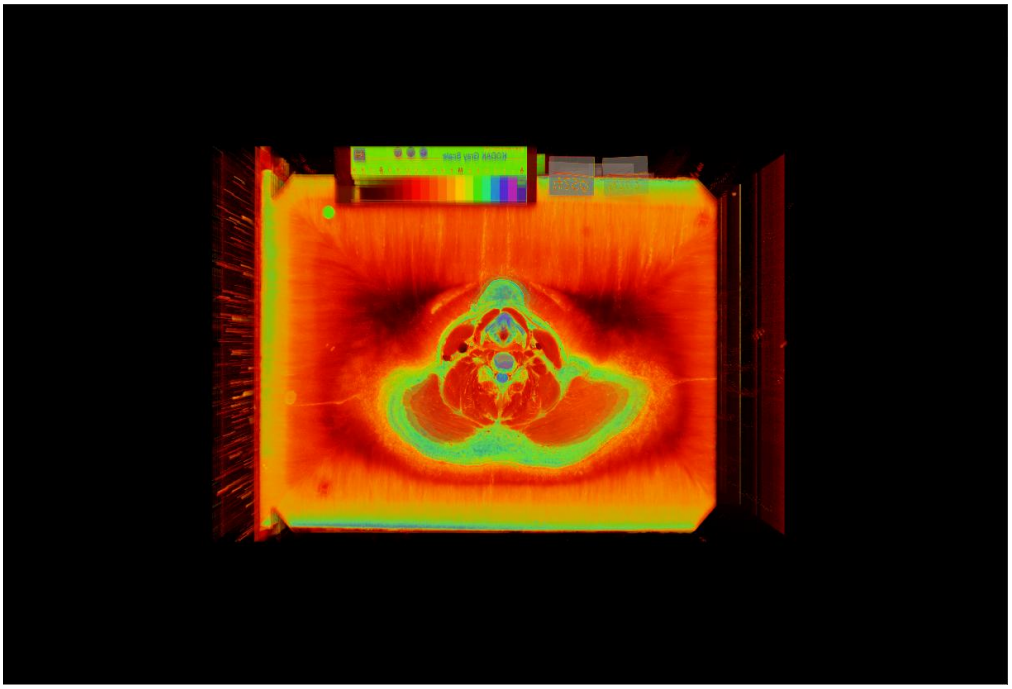

Figura 14 Imagem resultante da execução da aplicação

Na próxima subseção será discutido sobre a fase de operações sobre a imagem, onde serão expostas as operações que podem ser realizadas sobre o volume resultante da projeção com o objetivo de facilitar a visualização do objeto de estudo, assim como o funcionamento destas operações realizadas sobre o volume.

3.3.4 Operações sobre a Imagem

Como pode ser visto na Figura 14 ao se realizar o processo visualização a imagem resultante na projeção não é muito satisfatória, pois apenas nos mostra o objeto de estudo de um ponto de vista e contém muita informação que não é de interesse do usuário. Para isso existem operações que podem ser realizadas sobre o volume, a maioria das operações ocorre durante o processo de Ray Tracing e apenas a operação de corte ocorre na fase de manipulação dos dados. Todo o tratamento de eventos necessário para a execução das operações da aplicação é realizado pela biblioteca OpenGL, que é responsável por gerenciar os eventos de teclado e mouse durante a execução do aplicativo. Nessa subseção será descrita

quais as operações que podem ser realizadas pelo usuário, como elas são implementadas e o resultado da aplicação de cada um a das operações.

Corte – A captura dos dados do volume pode incluir na maioria das vezes informações do ambiente onde estas foram realizadas. Essas informações podem ser importantes para teste de desempenhos de aplicações, mas não são de interesse do usuário que pode ter a visão do objeto de estudo comprometida pela quantidade de informação excedente, como pode ser visto na Figura 14.

A função de corte (*crop*) tenta remover as informações de ambiente de maneira a permitir que o objeto de estudo do volume possa ser melhor visualizado. Nesta aplicação, como o objeto que será utilizado é fixo, também foi definido o tamanho fixo do corte. A operação de corte consiste em substituir os valores dos pixels que não pertencem ao volume por elementos neutros a operação aplicada por cada raio, neste caso substituindo por zero. A Figura 15 apresenta a operação de corte realizada sobre a Figura 14.

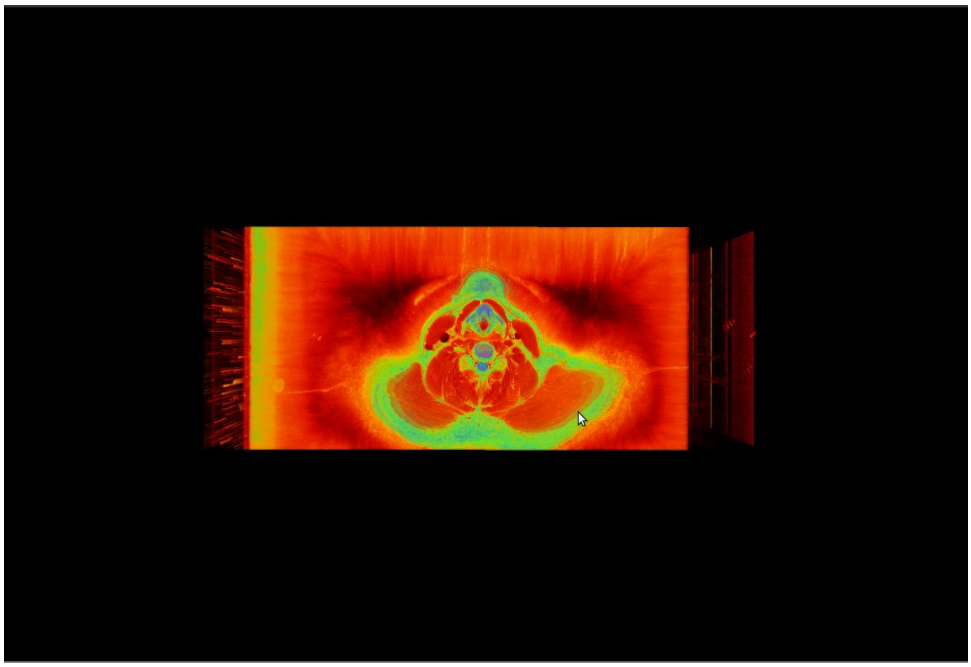

Figura 15 Resultado da operação de corte sobre a imagem

A função de corte pode ser utilizada ou não de acordo com a preferência do usuário, pois como foi discorrido anteriormente para efeitos de teste as informações de ambientes podem ser necessárias. A operação de corte ocorre sobre o vetor com os dados da imagem que existe na memória principal por isso é necessário também

liberar a memória da placa gráfica e transferir o novo conjunto de dados modificados para ela. Essa é a única operação que ocorre sobre os dados, as demais só são realizadas sobre a imagem resultante da projeção e ocorrem dentro da placa gráfica, ou são operações realizadas pela biblioteca OpenGL. Todas as imagens que serão utilizadas a partir de agora serão resultantes da operação de corte.

Limiarização – Devido a características da forma do objeto de estudo nem todas as informações de ambiente contidas no volume podem ser removidas sem que haja remoção de informações do objeto de estudo, com intuito de remover essas informações remanescentes do ambiente é utilizada a limiarização. Essa função remove valores da amostragem dos raios que estão abaixo do valor de limiar definido pelo usuário da aplicação. A Figura 16 mostra o resultado da aplicação da limiarização sobre a imagem.

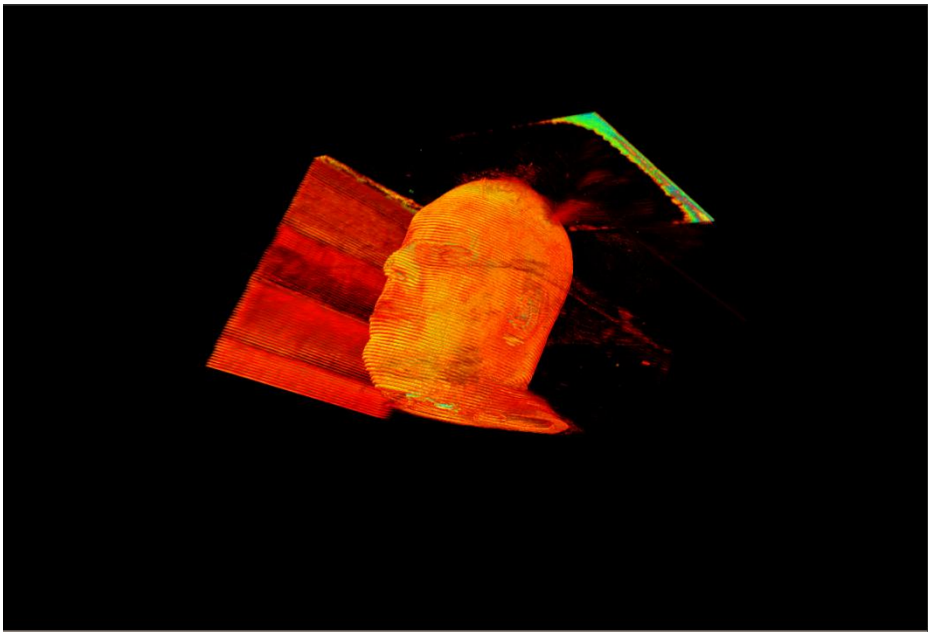

Figura 16 Exemplo de aplicação da operação de limiarização

Movimentação do volume – Uma funcionalidade presente nas aplicações de visualização volumétrica é permitir a movimentação do objeto de estudo. A movimentação aqui citada é a possibilidade de rotacionar o objeto de estudo de maneira a permitir sua visualização de diversos ângulos. Para rotacionar o volume o usuário deve mover o mouse clicando sobre o volume e arrastando para a posição desejada. A Figura 17 ilustra o movimento de rotação.

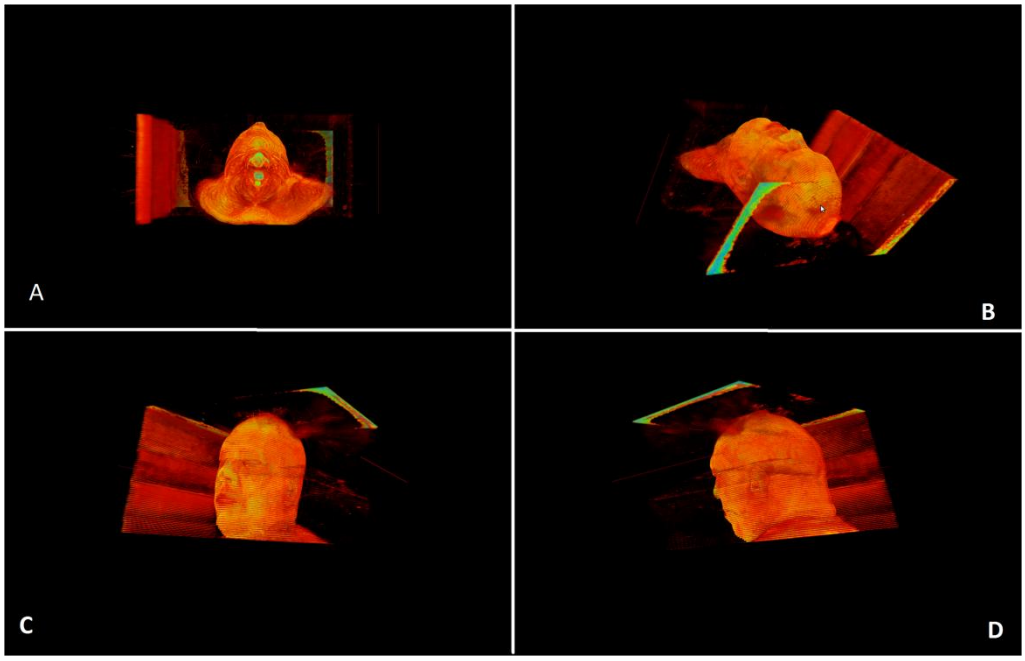

Figura 17 Diferentes ângulos de visualização

Cada uma das imagens ilustrada na Figura 17 apresenta a projeção do volume de dados visto sobre um ângulo diferente.

Filtro Linear – Essa funcionalidade é implementada pela biblioteca do CUDA e permite que o valor aplicado em cada pixel não seja apenas o amostrado pelo raio, mas sim uma o resultado de uma interpolação aplicada nos valores dos píxeis próximos de maneira a suavizar a imagem, a Figura 18 apresenta um comparativo entre imagens com e sem o filtro linear aplicado, a imagem A possui o filtro desativo enquanto a B foi capturada com utilização do filtro.

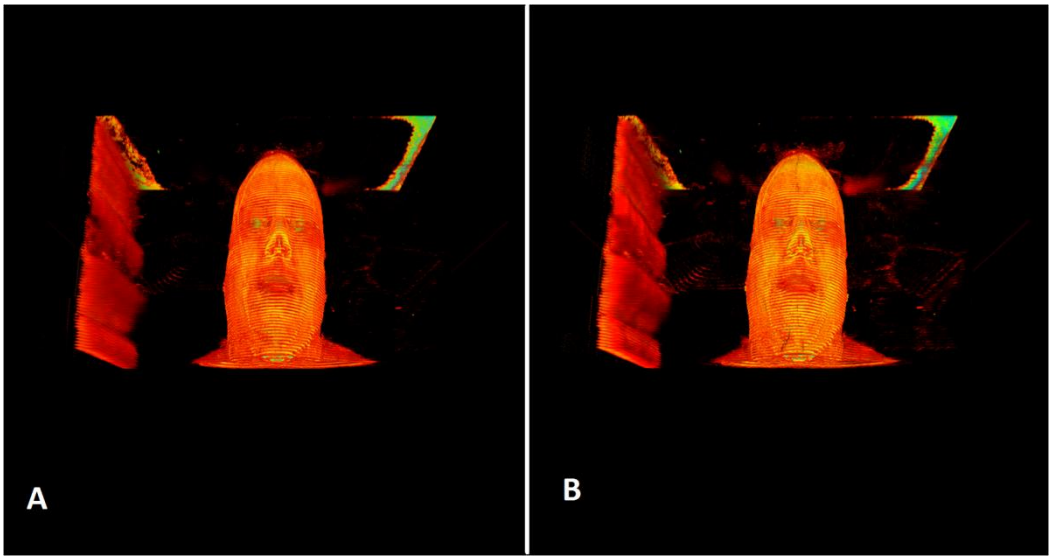

Figura 18 Exemplo da utilização do filtro linear

Escala de cor – É possível também alterar a escala de cores utilizadas, a operação é realizada simplesmente multiplicando a cor de cada amostra obtida em cada passo do raio pelo volume, a cor é alterada para uma frequência mais alta ou mais baixa na escala de cores de acordo com a necessidade do utilizador da aplicação. A Figura 19 apresenta a comparação de duas imagens do mesmo volume com escalas de cores diferentes.

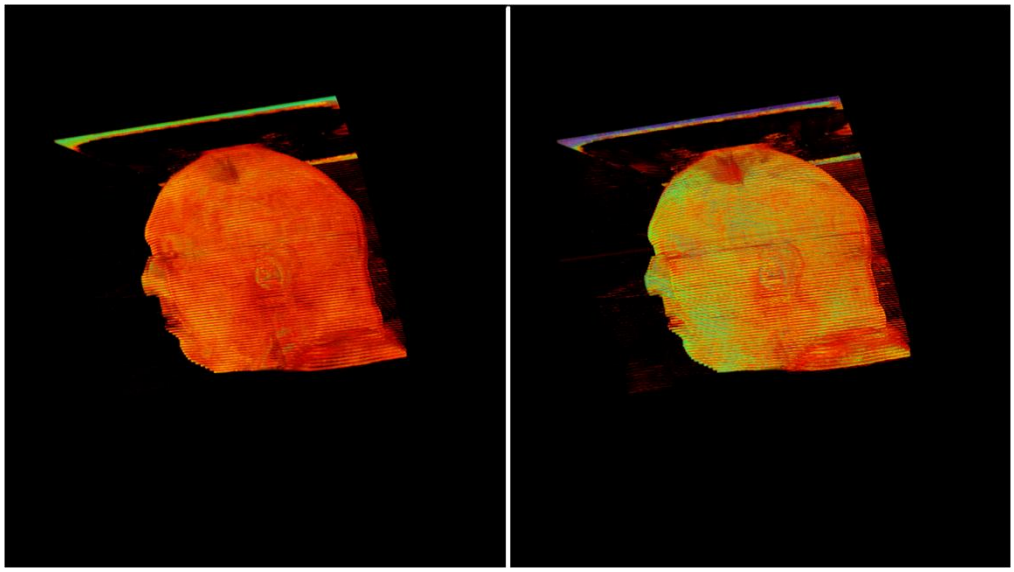

Figura 19 Exemplo da operação de Escala de Cor

Brilho – É possível se alterar o brilho da imagem permitindo uma melhor visualização, isso é realizado somando-se um valor a cor resultante de cada píxel a Figura 20 apresenta a aplicação da operação.

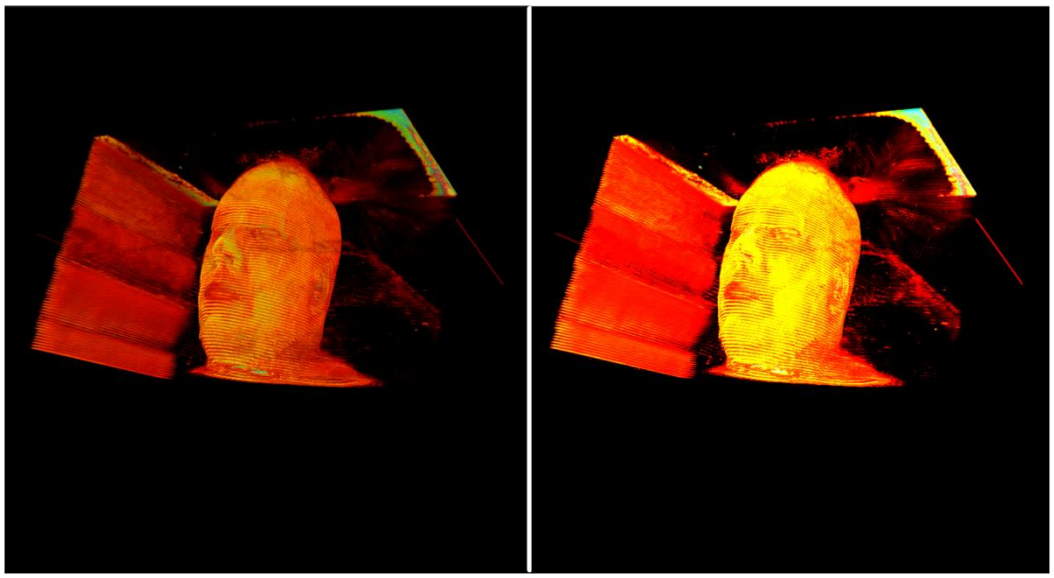

Figura 20 Exemplo da utilização da operação de brilho

Todas as operações apresentadas podem ser realizadas pelo usuário através do mouse utilizando o botão esquerdo do mouse no caso da rotação ou através do menu que pode ser acessado pressionando o botão direito do mouse em qualquer área da tela como ilustrado na Figura 21. Além dessas duas opções o usuário também pode utilizar teclas de atalhos definida e que são apresentadas quando ele ativa o menu.

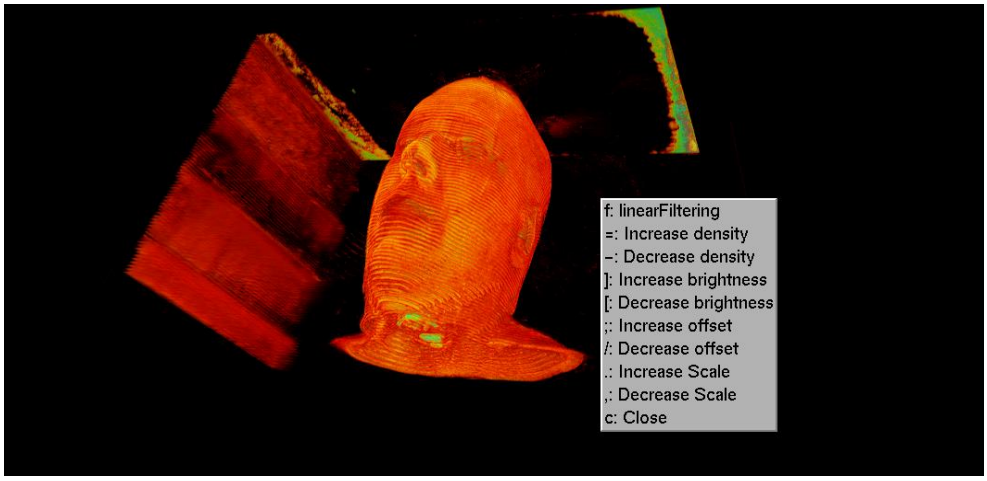

Figura 21 Menu da aplicação

3.4 Experimentos

A fim de avaliar o desempenho da execução do algoritmo Ray Trancing se utilizando a tecnologia a tecnologia CUDA. Foi realizado experimento onde se aferiu o número de frames por segundo (FPS) apresentado na tela, para cada valor diferente de distância entre as amostras dos raios. Como foi discutido anteriormente neste projeto o algoritmo de Ray Tracing percorre um caminho orientado pelos píxeis da área reservada para a projeção da imagem resultante em direção à base de dados. Para cada píxel, um raio é traçado em direção aos dados onde pontos amostrados em sua extensão determinam cor e opacidade a ser definida para o píxel. Neste projeto cada raio caminha pela base de dados realizando no máximo 5.000 passos (amostras). Dependendo da distância entre os passos o raio pode atravessar todo o volume sem realizar o número máximo de passos, isso ocasiona uma imagem com pouca qualidade, mas que pode ser gerada rapidamente. Em outro caso, com uma pequena distância entres os passos o resultado é uma imagem com alta qualidade e com uma taxa de FPS baixa devido ao grande número de cálculos necessário para se obter a imagem.

A Figura 22 apresenta uma comparação entre duas imagens resultantes da operação de corte e limiarização para ilustrar a diferença na qualidade da imagem com valores de distância entre amostras diferentes. A imagem A possui a distância entre amostras de 0,0005 com uma taxa de FPS igual a 2,1, enquanto a imagem B possui a distancia entre as amostras de 0,005 e a taxa de atualização de 17,2 fps.

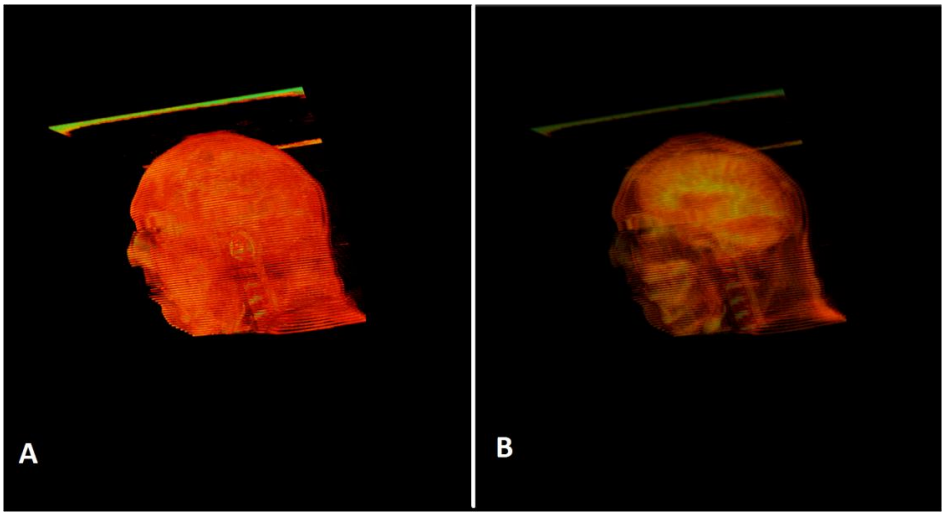

Figura 22 Exemplo de imagens com distância entre amostras diferentes: (a) distância entre amostras igual a 0,0005 e (b) distância entre amostras igual a 0,005

Para a realização desse experimento foram considerados as informação do ambiente contidos no volume, ou seja, não foi realizada a operação de corte sobre o volume. Entretanto foi aplicado o processo de limiarização com o valor de limiar igual a 0,25, essa operação foi aplicada com o objetivo de fazer com que o raio atravesse todo o volume, impedindo que ele pare de realizar amostras caso encontre um valor opaco nas informações de ambiente. Todas as medições foram realizadas com volume na mesma posição, de modo que a única diferença entre as imagens resultantes seja somente o valor da distância entre as amostras obtidas pelo raio, a Figura 23 exemplifica uma das imagens obtidas durante o experimento essa imagem foi gerada com valor da distância entre as amostras fixadas em 0,0009.

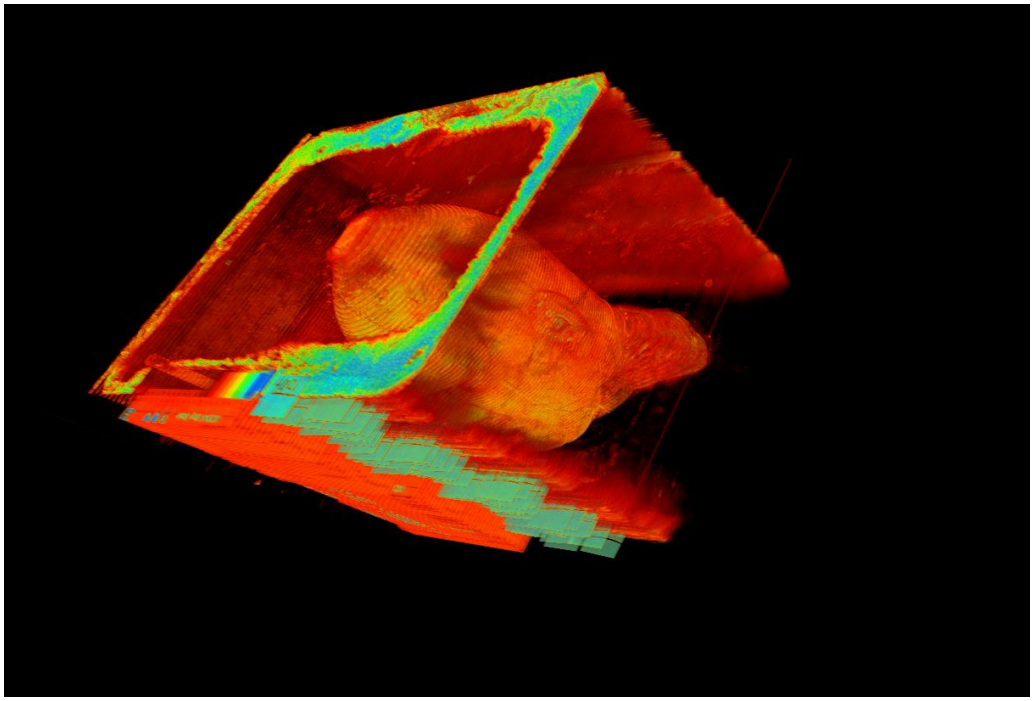

Figura 23 Imagem com distância entre amostras de 0,0009

A medição dos valores de FPS foi realizada com os valores das distâncias entres amostras variando entre 0,0005 a 0,01, o valor do FPS era obtido a cada variação de 0,0001 na distância das amostras, a tabela com todos os resultado obtidos no experimento pode ser vista no Apêndice A. Com os valores obtidos pelo experimento foi construído o gráfico apresentado na Figura 24.

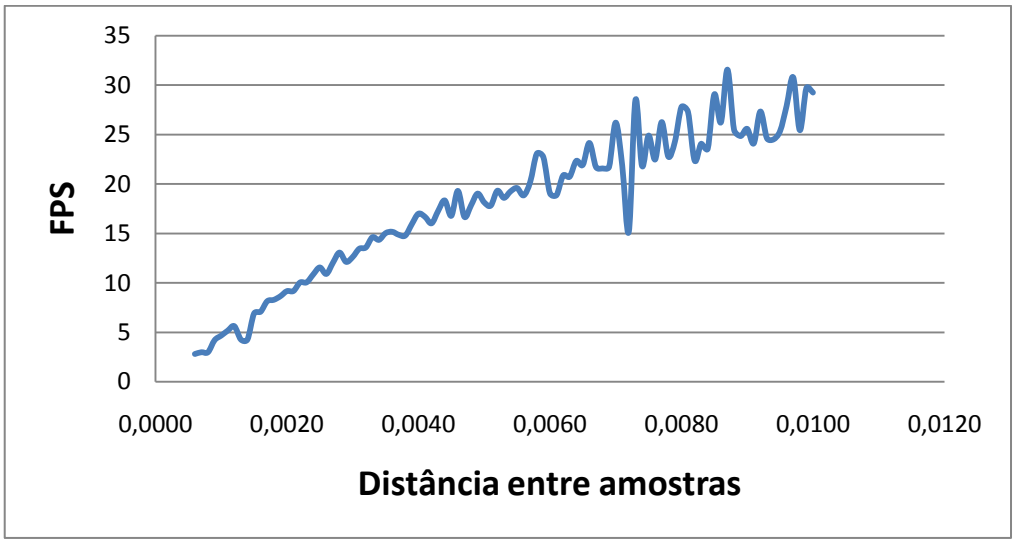

Figura 24 Resultados obtidos no comparativo entre FPS e distância entre amostras

A partir da análise do gráfico apresentado na Figura 24 é possível perceber que a relação existente entra o número de FPS e a distância entre as amostras segue uma tendência linear, mas não proporcional. Além de que a partir da distância entre as amostras com valor de 0,006 existe uma tendência dos valores se estabilizarem, considerando a faixa de valores amostrados. Essa tendência em estabilizar os valores é consequência de limitações de hardware relacionadas à gerência de *threads* e velocidade de leitura de memória.

A seleção do valor da distância entre as amostras, utilizado pela aplicação, deve levar em consideração a qualidade da imagem resultante desejada e o hardware disponível, pois com um valor baixo de distância entre amostras temos imagens de boa qualidade, mas que exige muito processamento para ser gerada. Essa quantidade de processamento necessário pode tornar o aplicativo lento e pode sobrecarregar o hardware. Durante a execução desse experimento foi observado que a aplicação por vezes foi finalizada pelo sistema operacional devido à alta temperatura da placa gráfica.

Levando-se em consideração todas as informações obtidas através do experimento foi definida a utilização de distância entre as amostras de 0,001 que gera imagens com excelente qualidade e possui uma taxa de FPS que permite a utilização da aplicação sem problemas. Esse valor proposto deve ser considerado para placas gráficas com as mesmas especificações da que foi utilizada neste projeto, outras placas gráficas terão desempenhos distintos dos observados neste experimento.

3.5 Considerações Finais

Neste capítulo foi apresentada a aplicação proposta no projeto deste trabalho, os dados utilizados, as funcionalidades implementadas, o resultado da aplicação e também foi apresentado o experimento realizado para analisar a relação existente entre a distância da amostragem realizado pelo raio que passa sobre a base de dados e a taxa de atualização da imagem. No próximo capítulo serão apresentadas as conclusões obtidas a partir da execução deste trabalho.

4
Conclusões

De acordo com o conteúdo neste documento, é possível concluir que este trabalho foi concretizado inicialmente se realizando um estudo sobre as técnicas de computação gráfica e sobre a técnica de Ray Tracing. Uma vez que se obteve a compreensão do problema proposto partiu-se para o estudo da tecnologia CUDA e para a busca de uma alternativa para se criar um algoritmo de Ray Tracing que possa tirar proveito da tecnologia de placas gráficas programáveis.

Como resultados práticos temos a criação da aplicação de visualização volumétrica de dados médicos que utiliza a tecnologia CUDA para a realização do processo de Ray Tracing, também foi obtido uma tabela comparativa entre a distância das amostras e a taxa de atualização da tela de visualização de maneira a ilustrar o desempenho da aplicação desenvolvida quando executada na placa gráfica utilizada no projeto.

No que diz respeito a limitações do projeto, não foi possível realizar um teste comparativo dos resultados obtidos entre o sistema desenvolvido neste trabalho e outro que utilize o mesmo algoritmo de Ray Tracing executado exclusivamente na CPU. Além disso, não foi possível a realização de experimentos variando outras variáveis como o número de *threads*, ou a execução da aplicação em outras placas com suporte a tecnologia CUDA.

Considero importante também mostrar algumas das limitações impostas pela tecnologia que foram observadas durante a execução do projeto. Temos como fator limitante o tamanho da memória disponível pela placa gráfica que nos impossibilitou de usar uma quantidade maior de dados, essa mesma limitação pode evitar o desenvolvimento de aplicações comerciais que utilizem a tecnologia. Além disso, ao analisar a declaração da NVIDIA de que uma aplicação que utilize a tecnologia CUDA possa ser executada em todas as placas que possuam suporte a tecnologia, deve se levar em consideração que uma aplicação desenvolvida para uma placa com memória de 1GB e que necessite de toda a memória disponível, como é o caso da aplicação apresentada neste projeto, não poderá ser executada em uma placa com memória de tamanho inferior mesmo as duas possuindo GPUs de

mesma família. Apesar das limitações que existem na tecnologia devemos considerar que a programação de placas gráficas está apenas no início e se mostra promissora no futuro e cada vez mais presente nos estudos realizados.

5 Referências Bibliográficas

Bressan, P. A. *Visualização volumétrica aplicada em aglomerados de computadores convencionais*. 2004. 104 p. Tese (Doutorado) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2004.

Carneiro, M. M.; Velho, L. Um estudo de algoritmos para visualização simultânea de dados volumétricos e superfícies poligonais. Rio de Janeiro: PUC-RJ, 2000. 54 p. Relatório técnico.

Drebin, R. A.; Carpenter L. & Hanrahan, P. *Volume rendering*. In: international conference on computer graphics and interactive techniques, 1988. Proceedings. New York: ACM Computer Graphics, 1988, v. 22, n. 4, p. 125-134.

Farias, R. & Bentes, C. Renderização Volumétrica Paralela de Dados Médicos Irregulares em Clusters de Pcs. In: CONGRESSO BRASILEIRO DE COMPUTAÇÃO, 2004, Itajaí. Anais do Workshop de Informática Aplicada à Saúde, 2004.

Halfhill, T. R. *Parallel Processing with CUDA*. Technical Report 01/28/08-01. Reed Electronic Group, 2008. Disponível em:
<http://www.nvidia.com/docs/IO/55972/220401_Reprint.pdf>. Acesso em: 20 nov. 2009

Kaufman, A. E. *3D Volume visualization*. Advances in Computer Graphics Vi, Images: Synthesis, Analysis, and interaction (Tutorials From Eurographics'90 Conf.) G. Garcia and I. Herman, Eds. Springer-Verlag, London, p. 175-203, 1991.

Kim, J. & Jaja, J. *Streaming model based volume ray casting implementation for Cell Broadband Engine*. Scientific Programming. v. 17, n 1-2, p. 173-184, jan. 2009.

Mccormick, B., Defanti, T. & Brown, M. *Visualization in Scientific Computing*. SIGBIO Newsl.v 10, n 1, p. 15-21, mar. 1988.

National Library of Medicine. Visible Human Project. [Mantido pelo National Institutes of Health] 2010. Disponibiliza informações sobre o projeto. Disponível em : <http://www.nlm.nih.gov/research/visible/index.html> Acessado em 02 jun 2010.

NVIDIA. *Compute Unified Device Architecture (CUDA) Programming Guide*, Version 2.0 edition. 2008. Disponível em:
<http://developer.download.nvidia.com/compute/cuda/2_0/docs/NVIDIA_CUDA_Programming_Guide_2.0.pdf>. Acesso em: 16 nov. 2009

Paiva, A. C.; Seixas, R. B. & Gattass, M. Introdução à visualização volumétrica. Rio de Janeiro, 1999. 106 f. Monografia (Especialização em Ciência da Computação) - Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Popov, S.; Günther, J.; Seidel, H. P. & Slusallek, P. *Stackless kd-tree traversal for high performance gpu ray tracing*. Computer Graphics Forum, v. 26, n. 3, p. 415-424, set. 2007.

Ronghua, L.; Zhigeng, P.; Krokos, M.; Mao, C.; Jiarui, B. & Cheng LI. *Fast Hardware-Accelerated Volume Rendering of CT Scan*. Display Technology, Journal of , v 4, n .4, p.431-436, dez. 2008

Stone, J. E., Saam, J., Hardy, D. J., Vandivort, K. L., Hwu, W. & Schulten, K. *High performance computation and interactive display of molecular orbitals on GPUs and multi-core CPUs*. Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units. ACM, New York, NY, 9-18, v. 383, p. 9-18, mar. 2009

Walters, J. P.; Balu, V.; Chaudhary, V.; Kofke, D. & Schultz, A. Accelerating Molecular Dynamics Simulations with GPUs. Proceedings of the 21st International Society for Computers and their Applications, Parallel and Distributed Computing and Communication Systems (ISCA-PDCCS'08), New Orleans, LA, 2008

Zuffo, M. K. *Um ambiente de programação de alto desempenho para visualização volumétrica*. 1997. 174 p. Tese (Doutorado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 1997.

6 Apêndices e Anexos

Apêndice A – Tabela com resultado dos experimento

A Tabela 1 apresenta o valor de frame por segundos obtidos para cada um dos valores de distância entre amostras. O valor da distância entre as amostras varia entre 0,0005 e 0,01.

Tabela 1 Valores de Frames por segundo por distância entre as amostras

Distância entre amostras	*Frames por Segundos*
0,0006	2,828
0,0007	3,008
0,0008	3,008
0,0009	4,221
0,0010	4,685
0,0011	5,189
0,0012	5,641
0,0013	4,290
0,0014	4,290
0,0015	6,917
0,0016	7,099
0,0017	8,161
0,0018	8,300
0,0019	8,662
0,0020	9,183
0,0021	9,195
0,0022	10,057
0,0023	10,069
0,0024	10,866
0,0025	11,580
0,0026	10,919
0,0027	12,053
0,0028	13,080

0,0029	12,137
0,0030	12,628
0,0031	13,467
0,0032	13,594
0,0033	14,632
0,0034	14,362
0,0035	15,038
0,0036	15,185
0,0037	14,879
0,0038	14,796
0,0039	15,954
0,0040	17,010
0,0041	16,691
0,0042	16,035
0,0043	17,291
0,0044	18,355
0,0045	16,792
0,0046	19,321
0,0047	16,683
0,0048	17,842
0,0049	19,043
0,0050	18,157
0,0051	17,825
0,0052	19,329
0,0053	18,608
0,0054	19,258
0,0055	19,604
0,0056	18,856
0,0057	20,216
0,0058	23,077
0,0059	22,729
0,0060	19,077
0,0061	18,882
0,0062	20,871
0,0063	20,746
0,0064	22,342
0,0065	21,943
0,0066	24,167
0,0067	21,727
0,0068	21,589
0,0069	21,741
0,0070	26,219

0,0071	21,849
0,0072	15,195
0,0073	28,487
0,0074	21,839
0,0075	24,910
0,0076	22,486
0,0077	26,274
0,0078	22,779
0,0079	24,272
0,0080	27,802
0,0081	27,296
0,0082	22,387
0,0083	24,078
0,0084	23,578
0,0085	29,072
0,0086	26,235
0,0087	31,575
0,0088	25,497
0,0089	24,855
0,0090	25,587
0,0091	24,113
0,0092	27,340
0,0093	24,638
0,0094	24,519
0,0095	25,356
0,0096	27,926
0,0097	30,795
0,0098	25,449
0,0099	29,704
0,0100	29,252