UNIVERSIDADE FEDERAL DE ALFENAS

MARCO ANTÔNIO BARBOSA FILHO

VISUALIZAÇÃO VOLUMÉTRICA EM UNIDADES DE PROCESSAMENTO GRÁFICO

Alfenas/MG
2017

MARCO ANTÔNIO BARBOSA FILHO

VISUALIZAÇÃO VOLUMÉTRICA EM UNIDADES DE PROCESSAMENTO GRÁFICO

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Alfenas - Unifal-MG como requisito para obtenção do título de Bacharel em Ciência da Computação.
Orientador: Prof. Dr. Paulo Alexandre Bressan.

Alfenas/MG
2017

MARCO ANTÔNIO BARBOSA FILHO

VISUALIZAÇÃO VOLUMÉTRICA EM UNIDADES DE PROCESSAMENTO GRÁFICO

A Banca examinadora abaixo assinada, aprova o Trabalho de Conclusão de Curso como parte dos requisitos para a obtenção do título de Bacharel em Ciência da Computação pela Universidade Federal de Alfenas.

Aprovada em:

Prof.: Dr. Paulo Alexandre Bressan
Instituição: UNIFAL-MG Assinatura:

Prof.: Daniel Borges Torrico Villarreal
Instituição: UNIFAL-MG Assinatura:

Prof.: Dr. Ricardo Menezes Salgado
Instituição: UNIFAL-MG Assinatura:

RESUMO

Visualização volumétrica consiste em um conjunto de técnicas capazes de extrair informações relevantes a partir de complexas bases de dados. Aplicações para visualização volumétrica são cada vez mais comuns, pois a partir de imagens 2D são gerados objetos em 3D, que permitem observar com mais detalhes o que está sendo estudado. A visualização volumétrica consiste no processamento digital de um conjunto de dados que resultará em um produto final, esse processamento é chamado de renderização. Devido ao aumento na demanda por tecnologias capazes de realizar renderizações de maneira eficiente, tanto na área científica quanto no entretenimento, foram criadas placas gráficas com grande poder computacional (GPUs - *Graphics Processing Units*). As placas gráficas que possuem a tecnologia CUDA (*Compute Unified Device Architecture*) oferecem uma extensão para a linguagem de programação C/C++, que proporciona e facilita o uso da computação paralela. Neste contexto, o trabalho a seguir se propôs a realizar modificações em uma aplicação que realiza a renderização volumétrica de dados médicos utilizando uma placa gráfica com tecnologia CUDA (*Compute Unified Device Architecture*), com o objetivo de realizar melhorias em relação ao desempenho e qualidade de visualização.

Palavras-chave: Visualização volumétrica. Renderização volumétrica. Placa gráfica.

ABSTRACT

Volumetric visualization comprises of a set of techniques able to extract relevant information from complex databases. Applications for volumetric visualization are common nowadays, since from 2D imagens, 3D objects are created, allowing the observer to see with more details what is being studied. Volumetric visualization consists in digital processing of a dataset that will result in a final product, and this processing is called rendering. Due to the increasing demand for technologies capable of performing rendering efficiently, both for scientific and entertainment purposes, graphic cards were built with high computational power (GPUs – Graphics Processing Units). Graphic cards possessing CUDA technology (Compute Unified Device Architecture) offer an extension for C/C++ programming language, providing and facilitating the use of parallel computation. In this context, the following study proposes to modify an application that performs volumetric rendering of medical data using a graphic card with CUDA (Compute Unified Device Architecture) technology, aiming for improvements regarding the performance and quality of visualization.

Keywords: Volumetric visualization. Volumetric rendering. Graphic card.

LISTA DE FIGURAS

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

2D	-	Duas dimensões ou bidimensional
3D	-	Três dimensões ou tridimensional
API	-	*Application Programming Interface*
CPU	-	*Central Processing Unit*
CUDA	-	*Compute Unified Device Architecture*
FPS	-	*Frames Per Second*
GIMP	-	*GNU Image Manipulation Program*
GNU	-	*GNU is Not Unix*
GPU	-	*Graphics Processing Unit*
GTK	-	*GIMP Toolkit*
LGPL	-	*Lesser General Public License*
NLM	-	*National Library of Medicine*
OpenGL	-	*Open Graphics Library*
PC	-	*Personal Computer*
PNG	-	*Portable Network Graphics*
RAM	-	*Randon Access Memory*
RGB	-	*Red, Green, Blue*
SIMD	-	*Single Instruction Multiple Data*
UTF8	-	*8-bit Unicode Transformation Format*

SUMÁRIO

1 INTRODUÇÃO

De acordo com Manssour e Cohen (2006) a computação gráfica é o conjunto de técnicas e algoritmos para a sintetização de imagens através do computador. Atualmente, é possível utilizá-la em diversas áreas do conhecimento, como por exemplo no desenvolvimento de automóveis ou na criação de ferramentas para o entretenimento. O crescente uso da computação gráfica proporcionou o desenvolvimento de tecnologias que facilitaram o surgimento de novas aplicações. Ainda de acordo com Manssour e Cohen (2006), através das facilidades oferecidas pelas bibliotecas gráficas, a programação está se tornando cada vez mais simples. A biblioteca gráfica OpenGL (*Open Graphics Library*) por exemplo, permite a criação de aplicações interativas e torna possível também gerar imagens de cenas 3D com um alto grau de realismo. Além disso, o uso da computação gráfica para a visualização volumétrica vem crescendo constantemente, gerando representações de dados volumétricos.

Segundo Bressan (2004), a visualização volumétrica consiste na apresentação de dados escalares e vetoriais em três ou mais dimensões, formado pelo conjunto de algoritmos, técnicas e metodologias, que sintetizam imagens a partir de dados organizados em forma de volume. Paiva, Seixas e Gattas (1999) afirmam que a visualização volumétrica e o processo de renderização são importantes para a interpretação de dados em diferentes áreas, como por exemplo na saúde em que os profissionais precisam analisar dados de experiências ou resultados de exames, tais como tomografias computadorizadas e ressonâncias magnéticas. Além disso, pode ser amplamente utilizada para meteorologia, astrofísica, química, engenharia mecânica, entre outras.

A renderização de volumes tem recebido bastante atenção da comunidade científica de acordo com Farias e Bentes (2004). Isso ocorre devido a sua importância no estudo 3D de dados médicos e científicos. Os algoritmos que realizam a renderização volumétrica tratam os dados como compostos de material semitransparente, assim, possibilitando a visualização de seu interior. Entretanto, o processo de renderização é dispendioso, até mesmo algoritmos otimizados levam tempo para executar uma aplicação em apenas um processador. Porém, como afirma Camargo (2010) a visualização volumétrica é um problema que está inserido no

conjunto de problemas paralelizáveis. Conjunto esse que tem se beneficiado com o surgimento de processadores com mais de um núcleo e com o surgimento de placas gráficas programáveis.

Placas gráficas são constituídas de diversos circuitos e elementos eletrônicos, entretanto, o seu principal componente é uma unidade de processamento responsável pela renderização de gráficos em tempo real, conhecida como GPU (*Graphics Processing Unit*). Placas gráficas eram compostas por GPUs (*Graphics Processing Units*) com funções fixas, construídas de maneira que pouco implementava recursos tridimensionais. Atualmente, as GPUs possuem processadores programáveis e também APIs (*Application Programming Interface*) bem construídas que tornam a programação mais simples. O principal uso de placas gráficas ainda é o entretenimento, porém sua tecnologia tem atraído diversos pesquisadores, já que esses podem executar seus experimentos utilizando computadores pessoais que possuem um menor custo.

A tecnologia CUDA (*Compute Unified Device Architecture*) está presente em algumas placas gráficas tornando mais simples a programação nestes componentes. Isso ocorre pois, de acordo com Halfhill (2008 apud Camargo, 2010), ela oferece ferramentas de desenvolvimento em C/C++, bibliotecas de funções e um mecanismo de abstração de hardware que esconde detalhes do hardware da GPU do desenvolvedor. O programador pode fazer uso dessa tecnologia sem a necessidade de aprender uma nova linguagem de programação, pois o CUDA oferece ferramentas que fazem a tradução da linguagem conhecida para o código que utiliza recursos da placa gráfica, com isso facilitando o modelo de programação.

É possível encontrar na literatura diversos trabalhos sobre visualização volumétrica. Como por exemplo, a aplicação desenvolvida por Moraes, Amorim e Martins (2010) denominada InVesalius3. A aplicação possui seu código aberto e é utilizada para a reconstrução de imagens geradas a partir de tomografias computadorizadas ou ressonâncias magnéticas. Seu foco é principalmente na área de prototipagem rápida, ensino, análises forenses e na área médica. Moraes, Amorim e Martins (2010) fazem uso da técnica de Ray Casting, muito semelhante ao Ray Tracing, juntamente com outras tecnologias para a execução da renderização. Entretanto, não é feito o uso da GPU, o que poderia resultar em uma melhora na velocidade de execução.

O trabalho de Farias e Bentes (2004) apresenta o algoritmo denominado

PZSweep. Esse algoritmo consiste na paralelização do algoritmo ZSweep com o objetivo de ser executado em um *cluster* de PCs. O algoritmo desenvolvido apresenta bons resultados, onde a aplicação demonstrou tempos de execução viáveis a medida que mais processadores são incluídos no sistema, porém, essa adição de máquinas ao *cluster* pode ter um alto custo financeiro.

Já o trabalho de Iescheck, Sluter e Dedecek (2008) utiliza o processo de renderização volumétrica para a visualização de propriedades relacionadas a fenômenos geocientíficos. O trabalho foi desenvolvido utilizando um conjunto de programas que não faz uso da técnica de Ray Tracing, apresentando resultados interessantes onde é possível visualizar propriedades físicas e químicas do solo, mas a visualização não apresenta muitos detalhes. O trabalho conclui que a visualização volumétrica de solos e de outros fenômenos representados de maneira tridimensional é de grande importância para diversas áreas da ciência e consiste em uma importante ferramenta de apoio ao processo de análise.

De maneira semelhante, Novaes et al. (2012) propõem um método para a renderização volumétrica de dados sísmicos com o intuito de realizar interpretações geológicas. Neste trabalho foi utilizada a técnicas de Ray Tracing, resultando na visualização de objetos sísmicos do tipo *geobody* de maneira eficiente e com mais detalhes se comparado ao trabalho de Iescheck, Sluter e Dedecek (2008). Entretanto, para a realização deste projeto foi utilizada uma máquina de grande porte, sendo de difícil acesso para grande parte dos usuários.

Já o trabalho de Rúbio (2003) apresenta a implementação do algoritmo Ray Casting para a visualização de tumores intracranianos em exames de tomografias computadorizas. O algoritmo foi implementado de maneira otimizada com o intuito de reduzir o tempo de processamento em computadores convencionais. A aplicação renderiza um objeto com boa qualidade de visualização em um tempo de execução satisfatório de acordo com Rúbio (2003). Porém, o trabalho foi desenvolvido antes do surgimento do CUDA, tecnologia que poderia contribuir com a melhora no desempenho da aplicação.

O projeto de Camargo (2010) foi desenvolvido utilizando um computador com processador Pentium® Core 2 Quad 2.83GHz, 8 gigabytes de memória RAM e placa gráfica NVIDIA GeForce GTX 285 com 1 gigabyte de memória.

Durante o processo de carregamento dos dados, foi utilizado apenas metade dos dados que compõem o volume. Essa decisão foi tomada devido a limitação de

memória da placa gráfica, que possui apenas 1 gigabyte, sendo que o conjunto de dados completo tem 1,8 gigabyte. Os dados carregados foram armazenados em um vetor unidimensional em que cada três elementos do vetor representam a cor de um pixel. Em seguida, é feita a alocação de memória na placa gráfica e a cópia dos dados para a mesma.

A técnica escolhida para possibilitar a visualização volumétrica foi o Ray Tracing. Essa técnica faz com que os raios lançados percorram todo o volume de dados. A medida que o raio vai percorrendo o volume, é feita uma amostragem de cada célula voxel encontrada. Essas amostragens são acumuladas, ao terminar de percorrer o conjunto de dados o valor encontrado pelo somatório dessas amostragens é atribuído ao pixel que será projetado na tela. A distância em que é realizada cada amostragem afeta a qualidade do objeto gerado, assim, quanto maior a distância, menor a qualidade da imagem. Entretanto, quanto menor a distância maior o processamento necessário para realizar a renderização.

A aplicação possui algumas operações que melhoram a qualidade de visualização do objeto em estudo. Assim que é feita a renderização, a imagem resultante apresenta muito conteúdo não interessante ao usuário. Pensando nisso, foi implementada a operação de corte. Muitas vezes o conjunto de dados utilizado pode incluir informações do ambiente onde foi gerado. Essas informações não possuem valor ao usuário. O objetivo da operação de corte é então remover as informações do ambiente, tornando a visualização mais clara. Para isso, a função de corte substitui os valores que não fazem parte do volume de dados por valores neutros, que nesse caso é zero.

Outra operação implementada é o processo de limiarização. Algumas informações de ambiente não podem ser removidas sem prejudicar o objeto que está sendo estudado. Portanto, o objetivo da operação é remover informações de ambiente evitando prejudicar o objeto. A limiarização remove valores da amostragem que estão abaixo de um determinado limiar definido pelo usuário. A Figura 1 apresenta o resultado obtido após a utilização da operação de limiarização.

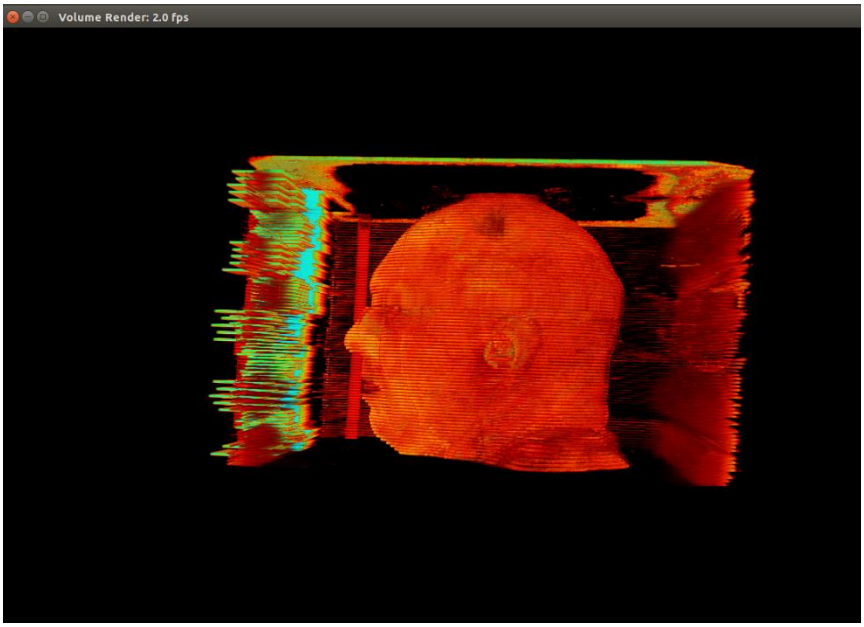

Figura 1 – Processo de limiarização implementado por Camargo (2010).

A operação de movimentação do volume permite ao usuário rotacionar o objeto renderizado, possibilitando a visualização de diferentes ângulos. A movimentação é uma operação presente em diversas aplicações de visualização volumétrica. Para que seja possível rotacionar o objeto, o usuário deve clicar sobre o volume e mover o mouse para a posição desejada. A Figura 2 mostra o objeto visto de frente após a sua movimentação.

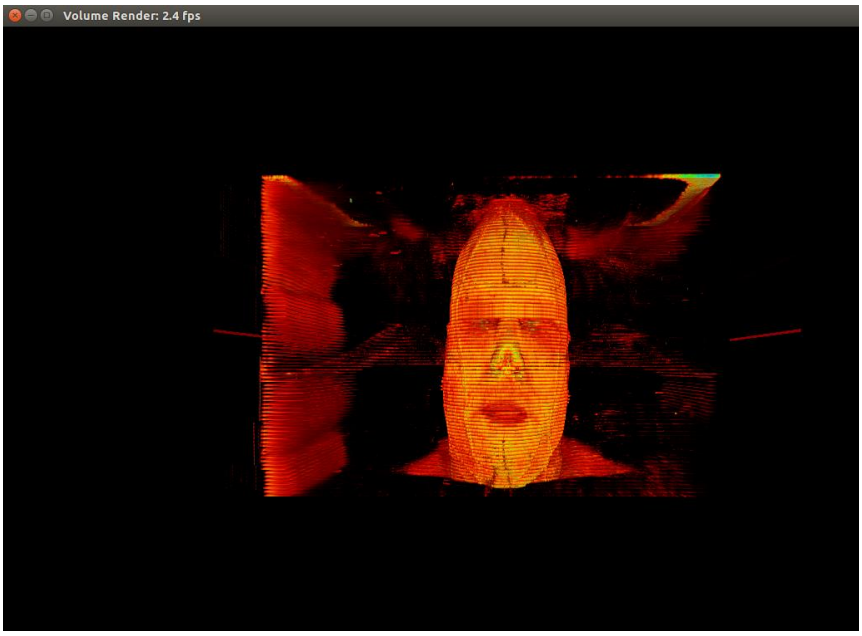

Figura 2 – Movimentação implementada por Camargo (2010).

A biblioteca do CUDA oferece algumas operações, dentre elas a filtragem linear que permite que o valor definido para cada pixel não seja apenas amostrado pelo raio. Com a utilização do filtro os valores dos pixels são definidos como sendo o resultado da interpolação entre os pixels próximos, com o objetivo de suavizar a imagem.

Além das operações citadas acima, é possível alterar a escala de cores do objeto resultante. Esse processo é feito através da multiplicação da cor de cada amostragem durante a passagem do raio pelo volume. Essa operação possibilita ao usuário alterar a cor para uma frequência mais ou mais baixa, de acordo com sua necessidade.

Por fim, também foi implementada a operação de brilho que permite ao usuário modificar o brilho da imagem. Isso é feito através da soma de um determinado valor a cor resultante de cada pixel.

Todas as operações citadas foram implementadas de maneira que fosse possível utilizá-las através de movimentos do mouse ou através de um menu que é apresentado ao se clicar com o botão direito do mouse em alguma área da janela de visualização, como mostra a Figura 3. Além disso, também é possível utilizar teclas de atalho, que são apresentadas junto ao menu.

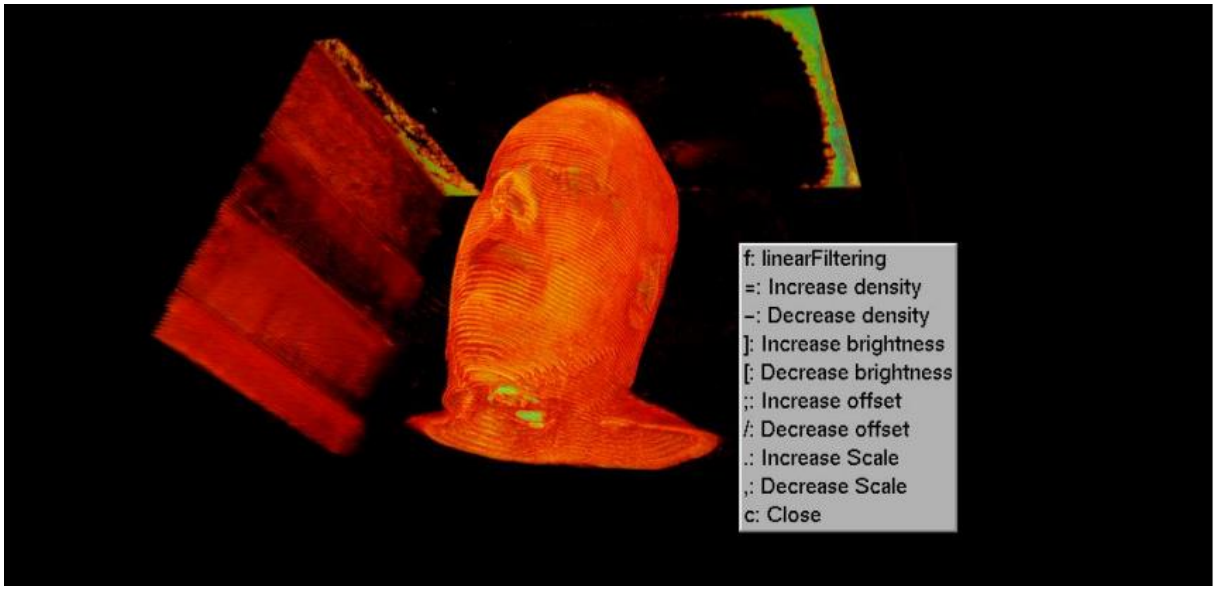

Figura 3 – Menu da aplicação.
Fonte: CAMARGO (2010, p. 56).

Neste contexto, este trabalho propõe modificações na aplicação desenvolvida por Camargo (2010), que realiza a renderização volumétrica de uma base de dados do projeto Visible Human, utilizando a tecnologia CUDA da placa gráfica, com tempo

de execução viável e de maneira acessível em relação ao seu preço. O objetivo do projeto é efetuar melhorias na aplicação em termos de desempenho e também em termos de qualidade de visualização.

Este trabalho está organizado da seguinte maneira: no Capítulo 2, será abordada a fundamentação teórica necessária para o entendimento do tema. No Capítulo 3, é apresentado o desenvolvimento do projeto e os resultados alcançados. Por fim, no Capítulo 4, é apresentada a conclusão sobre este trabalho.

1.1 OBJETIVOS

1.1.1 Gerais

Modificar a aplicação de visualização volumétrica utilizando a base de dados do projeto Visible Human desenvolvida por Camargo (2010) a fim de realizar melhorias em relação ao desempenho e também em termos de qualidade da imagem gerada.

1.1.2 Específicos

- Estudar fundamentos da visualização volumétrica;
- Estudar a base de dados do projeto Visible Human;
- Estudar a tecnologia CUDA;
- Estudar a técnica de Ray Tracing;
- Estudar a aplicação escolhida;
- Realizar as modificações na aplicação escolhida;
- Analisar os resultados obtidos após as modificações.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 VISUALIZAÇÃO VOLUMÉTRICA

De acordo com Paiva, Seixas e Gattass (1999), a visualização é um termo que se refere aos métodos que possibilitam a extração de informações relevantes a partir de complexos conjuntos de dados. É chamada de visualização científica quando os dados representam fenômenos complexos e o objetivo é a extração de informações científicas relevantes. Sendo a visualização volumétrica uma subárea da visualização científica. Segundo Manssour e Freitas (2002) a visualização volumétrica surgiu como um conjunto de técnicas de computação gráfica utilizadas para a apresentação de informações de volumes de dados complexos. O desenvolvimento de aplicações tem como objetivo fornecer aos usuários ferramentas que permitam diferentes maneiras de estudar, analisar e entender o interior de volumes de dados multidimensionais de maneira fácil e rápida. Camargo (2010) sugere que a visualização volumétrica pode ser apresentada como o processo em que informações abstratas de uma cena são transformadas em imagem.

Segundo Bressan (2004), o volume de dados é formado por um conjunto de voxels organizados espacialmente, sendo o voxel a unidade básica de um elemento volumétrico. O voxel é descrito por uma posição espacial e está relacionado a uma ou mais grandezas numéricas que representam propriedades do experimento. Podendo representar diferentes tipos de dados, como por exemplo: densidade, pressão, temperatura, carga eletrostática e tensão. Paiva, Seixas e Gattass (1999, p. 2) definem voxel como: “paralelepípedos, fortemente agrupados, formados pela divisão do espaço do objeto através de um conjunto de planos paralelos aos eixos principais desse espaço”.

Outro conceito importante para a visualização volumétrica é a célula voxel, apresentada na Figura 4, que consiste em um conjunto de voxels adjacentes conectados segundo uma topologia, definindo um espaço fechado. A forma mais comum de se representar uma célula voxels é através de um cubo, onde cada um dos oito vértices é definido por um voxel. Porém, é possível encontrar trabalhos na literatura que utilizam células voxel como tetraedros e pentaedros.

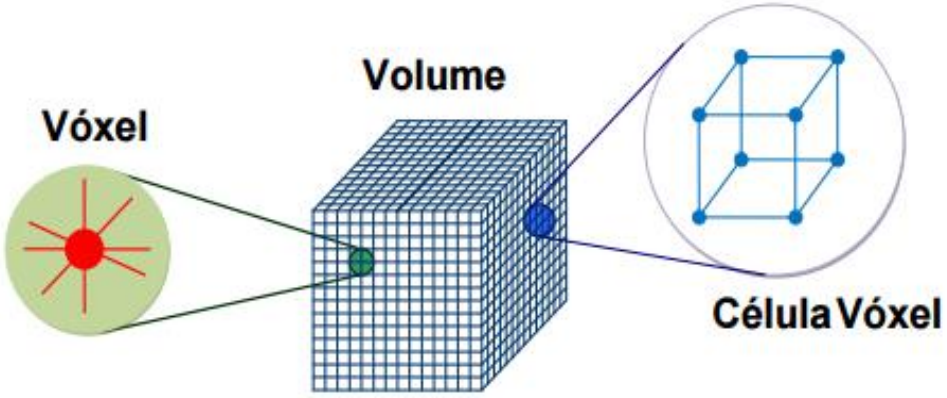

Figura 4 – Voxel, volume e célula voxel.
Fonte: CAMARGO (2010, p. 32).

2.1.1 *Pipeline* de Visualização

Carneiro e Velho (2000) apresentam em seu trabalho o *pipeline* de visualização, dividido em quatro etapas, sendo elas: aquisição, classificação, iluminação e projeção como mostra a Figura 5. Carneiro e Velho (2000) ainda afirmam que essas etapas são encontradas na maioria dos trabalhos relacionados ao processo de visualização volumétrica.

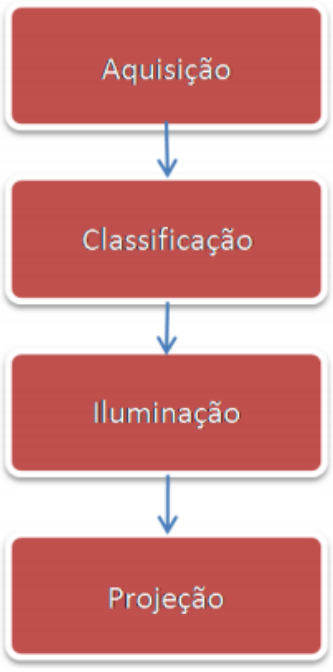

Figura 5 – *Pipeline* de visualização.
Fonte: CAMARGO (2010, p. 34).

O processo de aquisição geralmente é executado por meio de equipamentos de sensoriamento apropriados, como por exemplo: tomógrafos, sismógrafos, etc. Assim, o dado volumétrico adquirido está em seu estado bruto.

A classificação é a etapa mais difícil de ser realizada pelo usuário de um sistema de visualização volumétrica. Geralmente o usuário não tem conhecimentos técnicos sobre o processo de visualização, entretanto ele deve possuir um conhecimento prévio do objeto em estudo. O objetivo da classificação é identificar estruturas internas do volume. Assim, o usuário poderá por exemplo isolar determinadas estruturas que deseja visualizar, como por exemplo na visualização de uma tomografia em que o usuário poderia visualizar apenas tecidos ou apenas os ossos. Na utilização de algoritmos de renderização volumétrica direta, cujos dados são representados por grandezas escalares, é necessário definir a relação dos valores do volume com as cores que serão apresentadas. Já no caso da utilização renderização por isosuperfícies é necessário determinar quais os valores de limiarização serão poligonizados e apresentados.

O processo de iluminação é utilizado para criar a ilusão de profundidade, realçar bordas e algumas características do volume. Geralmente, os algoritmos de visualização volumétrica utilizam modelos de iluminação mais simples e que mesmo assim apresentam bons resultados, principalmente quando utilizados em imagens médicas. A utilização de modelos mais complexos pode proporcionar um grau maior de realismo, entretanto, se utilizado em dados médicos podem atrapalhar sua compreensão.

A etapa de projeção já recebe o volume de dados iluminado e realiza o processo de projetar os voxels sobre o plano da imagem, ou seja, é a etapa em que os dados são apresentados na tela de visualização, compondo a imagem a ser visualizada. É durante essa etapa que o usuário pode determinar o formato da visualização mais apropriado, definindo a área que deseja visualizar ou o objeto que quer rotacionar.

2.1.2 Classificação

Os dados volumétricos podem ser classificados de duas maneiras diferentes

de acordo com sua origem. Podem ser dados de simulação, que foram construídos a partir de algum modelo matemático. Ou dados de aquisição, que são resultantes da conversão de objetos observados da natureza.

De acordo com Manssour e Freitas (2002), existentem diversos termos e expressões para caracterizar diferentes técnicas utilizadas na visualização volumétrica. Entretanto, as duas principais categorias são a renderização volumétrica por isosuperfícies e a renderização volumétrica direta. Assim, Camargo (2010) afirma que a diferença principal entre as técnicas está no tipo do elemento central utilizado na renderização. Já que a renderização por isosuperfícies desenha superfícies geométricas através de primitivas gráficas, enquanto a renderização direta gera a imagem diretamente a partir do volume de dados. Neste trabalho, será considerada apenas a renderização volumétrica direta, pois foi a técnica utilizada na aplicação modificada.

2.1.3 Renderização Volumétrica Direta

Os algoritmos de renderização volumétrica direta tem como característica não fazer uso de representações intermediárias na construção de suas imagens, levando a um alto custo computacional. Lima (2005) afirma que a renderização volumétrica direta possibilita a visualização de todo o conjunto de dados.

De acordo com Bressan (2004), na técnica de renderização direta são traçados raios entre o observador e o objeto. Um raio é traçado para cada pixel presente na tela de visualização, onde é apresentada a imagem resultante. Os raios são lançados a partir do observador, diferente do modo como acontece na natureza, isso ocorre para que recursos computacionais sejam poupados. Se os raios fossem lançados de maneira semelhante a natureza, o processamento se tornaria inviável, pois a maior parte dos raios de luz que saem do objeto não chegam ao observador. O volume de dados é amostrado em intervalos regulares por toda a extensão de cada um dos raios, onde são gerados valores até que alguma condição seja satisfeita ou até que o raio saia do espaço do objeto. A cor do pixel é então definida pelo valor encontrado pela função ao percorrer o raio.

Os algoritmos de renderização volumétrica direta podem ser classificados de

duas maneiras em relação a sua composição, objeto-imagem e imagem-objeto. A composição objeto-imagem projeta voxels individuais no plano da imagem. Já a composição imagem-objeto lança raios do plano da imagem até o volume de dados. Através da composição imagem-objeto é possível interromper antecipadamente o processo de renderização se um determinado limiar for atingido. Já a composição objeto-imagem pode facilitar o processo, mas não possibilita a interrupção do processo de forma antecipada.

Algoritmos de renderização direta necessitam de uma função de transferência que realize a transformação de valores de voxel em valores de opacidade e outros que irão representar as características do objeto, como por exemplo: iluminação, cor, reflexo, etc. Essa função de transferência pode ser definida como uma tabela, permitindo uma maior flexibilidade da técnica de acordo com Inácio (2009).

Por esse motivo a renderização volumétrica faz uso constante de técnicas de interpolação numérica. Geralmente, a função de interpolação é tridimensional, porém, devido ao custo computacional, existem algoritmos que fazem uso da interpolação bidimensional. Manssour e Freitas (2002) afirmam que as imagens produzidas são de alta qualidade, já que todos os voxels podem ser utilizados na composição das imagens, possibilitando assim a visualização do interior do objeto em estudo.

2.1.3.1 Ray Tracing

O algoritmo de Ray Tracing consiste de maneira resumida em disparar um raio de cada pixel da janela de visualização em direção ao volume de dados, onde são amostrados pontos que determinam a cor e opacidade do objeto, Figura 6. De acordo com Bressan (2004), o Ray Tracing considera apenas a luz ambiente. Assim, os atributos do objeto são definidos apenas pelos valores amostrados, não sendo consideras a luz difusa, luz especular e nem reflexões múltiplas.

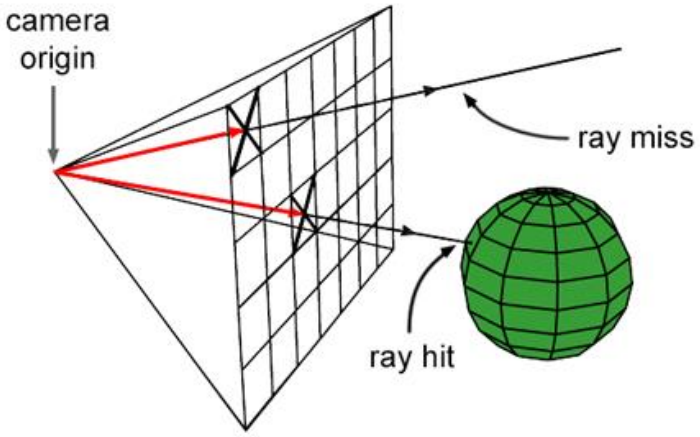

Figura 6 – Ilustração da técnica de Ray Tracing.

É possível definir diferentes funções para se aplicar quando o raio encontra um ponto, essas abordagens mudam dependendo das características da aplicação. Algumas implementações já são bastante conhecidas como por exemplo: primeiro valor, média das amostras, máxima intensidade e acumulativa, como apresenta a Figura 7. É possível compreender facilmente cada implementação analisando os valores dos voxels ao longo do raio.

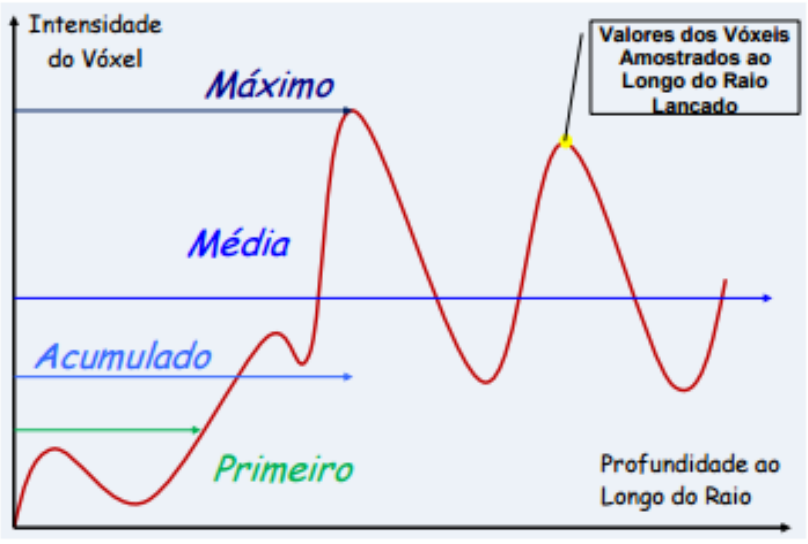

Figura 7 – Funções do Ray Tracing.
Fonte: Bressan (2004, p. 22).

A implementação do primeiro valor considera o valor do primeiro voxel encontrado numa dada faixa de valores, sendo esse o valor do pixel. A média das amostras é calculada utilizando o caminho completo do raio. A implementação da

máxima intensidade determina como o valor do pixel o maior valor encontrado no caminho percorrido pelo raio. E por fim, a acumulativa leva em consideração a opacidade atribuída a cada voxel e termina quando a cor do pixel é totalmente opaca ou o raio atravessa o volume de dados. A implementação acumulativa é a mais importante entre elas, pois apresenta melhor as propriedades internas da base de dados. A possibilidade de um término recente, ou seja, determinar a cor de um pixel sem a necessidade de percorrer todo o volume de dados é um poderoso método de aceleração para o Ray Tracing. Esse método consiste em finalizar a amostragem em um raio quando a opacidade acumulada alcançar um determinado valor.

2.1.3.2 Shear-Warp

O algoritmo de Shear-Warp trata de transformar a base de dados volumétrica de maneira a simplificar a etapa de projeção no *pipeline* de visualização. Essa simplificação é feita através do processo de cisalhamento nas fatias do volume de maneira que os raios de visão passem a estar perpendiculares as fatias. Esse processo permite que os dados sejam acessados na ordem de armazenamento, o que torna mais fácil a projeção das fatias. O cisalhamento é apenas uma transformação geométrica que tem como objetivo transladar as fatias do volume, sendo assim, um método que não emprega grande poder computacional, de acordo com Carneiro e Velho (2000). Através desse procedimento é gerada uma imagem intermediária que deve ser corrigida (transformação *warp*), como mostra a Figura 8.

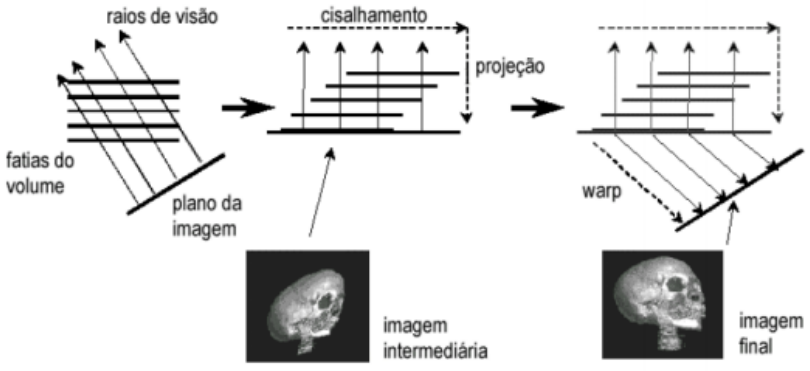

Figura 8 – Técnica Shear-Warp.
Fonte: CARNEIRO (2000, p. 35).

A principal vantagem do algoritmo Shear-Warp é que as dimensões da imagem intermediária são iguais ao tamanho do volume e do fator de cisalhamento. Portanto, a resolução da imagem final não influencia o tamanho da imagem intermediária. Tendo em vista que as dimensões do volume de dados são geralmente menores que a resolução final da imagem, o Shear-Warp acaba possuindo uma grande vantagem se comparado ao algoritmo de Ray Tracing.

2.1.3.3 Splatting

O algoritmo Splatting é aplicado no espaço dos objetos e realiza a projeção de cada voxel da base de dados no plano da imagem. De acordo com Carneiro e Velho (2000) o objetivo do algoritmo é “arremessar” cada voxel sobre o plano da imagem, o que faz com que o voxel deixe uma “marca” característica, sendo essa a origem no nome do algoritmo. A projeção utilizada no Splatting é normalmente realizada a partir dos voxels mais próximos do observador até o voxel mais distante.

A primeira decisão a ser tomada ao aplicar o algoritmo de Splatting é determinar em que ordem a base de dados será percorrida. Essa decisão é fundamental para o cálculo adequado da visibilidade, já que a ordem correta da projeção permite determinar de maneira correta as opacidades de cada voxel, de forma semelhante ao realizado no algoritmo de Ray Tracing. Da mesma maneira como ocorre nos outros algoritmos, o valor de densidade de cada voxel é calculado de acordo com as funções de transferências de cor e opacidade, sendo iluminado com o uso de uma técnica de estimativa normal através do gradiente.

O algoritmo de Splatting pode ser facilmente paralelizado, pois a projeção de cada voxel não leva necessariamente o restante do volume em consideração. Entretanto, a ordem correta da projeção deve ser mantida.

2.2 CUDA

O crescimento da demanda por tecnologias capazes de lidar com imagens de

alta resolução, principalmente no entretenimento, proporcionou o desenvolvimento de placas gráficas com alto poder computacional. Diferentemente das CPUs (*Central Processing Unit*), as placas gráficas apresentam milhares de núcleos desenvolvidos para lidar com múltiplas tarefas simultaneamente. Enquanto as CPUs (*Central Processing Unit*) possuem alguns núcleos otimizados para o processamento sequencial. A Figura 9 retrata a diferença na quantidade de núcleos entre a CPU e a GPU.

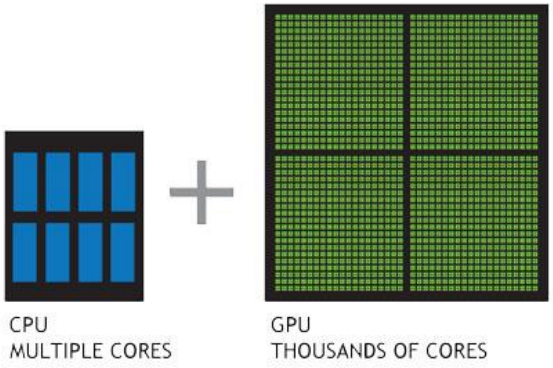

Figura 9 – Comparação entre a quantidade de núcleos entre CPU e GPU.
Fonte: NVIDIA (http://www.nvidia.com.br/object/what-is-gpu-computing-br.html).

As GPUs presentes nas placas gráficas são desenvolvidas para trabalhar de maneira paralela e realizar diversos cálculos com ponto flutuante, o que faz com que elas sejam ótimas para realizar o processo de renderização durante a visualização volumétrica, segundo Gomes (2013).

As GPUs oferecem melhor desempenho quando podem ser utilizadas em problemas em que a computação é feita utilizando a arquitetura SIMD (*Single Instruction Multiple Data*), onde a mesma instrução é aplicada para diferentes conjuntos de dados, de maneira paralela.

Gaioso et al. (2013) afirmam que as GPUs apresentam um maior poder computacional para o tratamento de imagens se comparadas às CPUs (*Central Processing Unit*), isso ocorre devido a sua arquitetura altamente especializada e paralelizada, diferentemente das CPUs que foram projetadas para a execuções sequenciais. As CPUs são muito mais generalistas, sendo capazes de realizar qualquer tipo de cálculo de processamento, até mesmo os gráficos, entretanto de maneira mais lenta. Já as GPUs são responsáveis apenas por lidar com o processamento gráfico, realizando cálculos de processamento de vetores, texturas e

desenhos que compõem as imagens na tela, de maneira bastante eficiente. Como mostra a Figura 10, a GPU é responsável por realizar o processamento intensivo de maneira paralela de uma pequena parte do código, enquanto a CPU executa o restante de maneira sequencial.

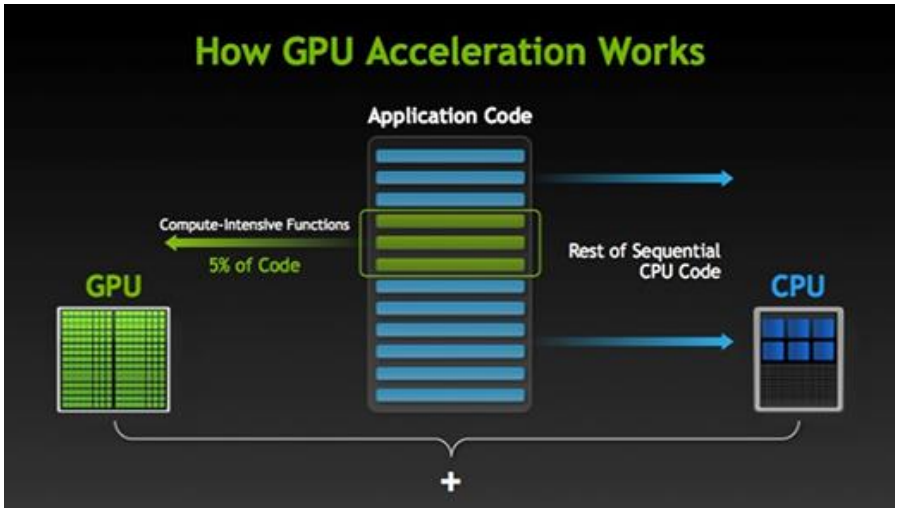

Figura 10 – Funcionamento da placa gráfica.
Fonte: NVIDIA (http://www.nvidia.com.br/object/what-is-gpu-computing-br.html).

Devido a esse maior poder computacional, pesquisadores passaram a se interessar na utilização de placas de gráficas. Entretanto, para que fosse possível utilizá-las em suas soluções era necessário que fosse feita uma transformação de seus problemas em problemas de computação gráfica. Além disso, era necessário aprender uma linguagem de programação específica para que fosse possível fazer uso das GPUs. Pensando nisso, em 2006 a NVIDIA apresentou o CUDA, uma solução que permitiria a programação utilizando GPUs de maneira muito mais simples, também conhecida como computação convencional em GPUs.

Segundo Vasconcelos, Carvalho e Gattass (2010), o objetivo do CUDA é fornecer um ambiente de desenvolvimento para GPU, permitindo a execução de uma grande quantidade de *threads* de uma maneira mais simplificada e paralela, com uma linguagem de programação semelhante à linguagem C.

De acordo com Camargo (2010) a vantagem de se utilizar CUDA é a abstração fornecida, pois a tecnologia “esconde” do programador detalhes do *hardware* onde o programa está sendo executado, tornando a programação muito mais simples, pois assim o programador não precisa entender exatamente a estrutura do *hardware* que está em uso. A abstração fornecida pelo CUDA é a mesma oferecida por

processadores, onde o usuário só precisa ter conhecimento de quais são as funções que o *hardware* suporta, sem a necessidade de saber como elas estão implementadas. Além disso, a abstração permite a portabilidade dos códigos desenvolvidos, pois mesmo que a NVIDIA futuramente altere seu *hardware* ou a maneira como as instruções são executadas, os códigos não precisarão ser reescritos para serem executados.

Essa abstração é aplicada através de dois dispositivos conhecidos como *host* e *device*, que representam a CPU e a GPU respectivamente, cada um responsável por diferentes tarefas. O *host* funciona como um processador de controle, responsável por iniciar variáveis, definir parâmetros e executar as partes seriais do código. Já o *device* é formado por conjuntos de multiprocessadores, sendo responsável por lidar com as partes que podem ser executadas de maneira paralela. As funções que são executadas pela GPU recebem o nome de *Kernel*. As funções *Kernel* são executadas N vezes paralelamente em N *threads*, diferentemente da execução sequencial feita pela CPU.

2.3 BASE DE DADOS

As imagens utilizadas na aplicação escolhida foram obtidas através do projeto Visible Human. O projeto foi idealizado em 1986 pela NLM (*National Library of Medicine*), nos Estados Unidos da América. O objetivo do projeto Visible Human é criar uma base de dados de imagens digitais de dois cadáveres, sendo um masculino e o outro feminino, obtidas através de tomografias computadorizadas e ressonâncias magnéticas. As imagens são disponibilizadas para serem utilizadas como referência para o estudo da anatomia humana, para testes de algoritmos de imagens médicas e também para pesquisas.

Para a aplicação foram escolhidas as imagens do cadáver masculino, obtidas em 1994 através da secção do corpo criogenizado. As imagens possuem 2048 pixels por 1216 pixels, onde cada pixel representa 0,33 milímetros da imagem. A Figura 11 apresenta uma das imagens que compõem a base de dados, resultante da tomografia computadorizada.

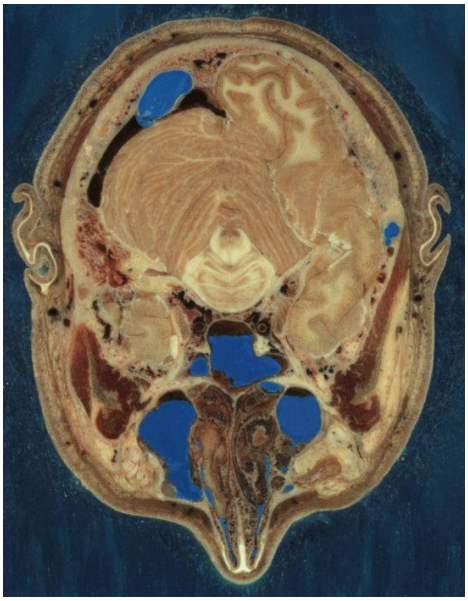

Figura 11 – Imagem resultante da tomografia computadorizada.
Fonte: PROJECT VISIBLE HUMAN (https://www.nlm.nih.gov/research/visible/visible_human.html).

A cor é definida por 24 bits, um byte para cada componente do padrão RGB. O projeto Visible Human oferece também as imagens da seguinte maneira:

- Imagens originais sem perda, armazenadas a partir de tomografias computadorizadas e de ressonâncias magnéticas;
- Imagens obtidas a partir da digitalização das imagens do corpo criogenizado, em formato RGB;
- Imagens originais sem perda, diretamente legíveis no formato PNG.

As imagens escolhidas foram armazenadas em formato RAW para que não houvesse perda de dados durante seu armazenamento. O nome RAW vem do inglês cru, que está relacionado ao modo como as imagens são armazenadas. Esse tipo de arquivo mantém os dados de maneira bruta, não aplicando nenhum tipo de compressão de dados, que causaria perda de informação. Dessa maneira, os dados contidos nesse arquivo representam a imagem digital de maneira fiel. Os arquivos em formato RAW são também chamados de negativos digitais, em comparação ao modo de armazenamento das câmeras analógicas.

Os arquivos do tipo RAW não possuem em seu cabeçalho informações sobre a imagem, como é comum de se encontrar em imagens digitais armazenadas em outros formatos, como por exemplo em imagens do tipo PNG que informam sobre o

tamanho da imagem e o padrão de cores utilizados. Essas informações vêm definidas em um outro arquivo, que no caso do projeto Visible Human são disponibilizadas em um arquivo de texto, onde são descritas informações sobre o tipo de armazenamento, o padrão de cores, o tamanho de cada imagem e o tamanho de cada pixel. A Figura 12 mostra a renderização do cadáver masculino a partir da base de dados cedida pelo projeto Visible Human.

Figura 12 – Renderização do cadáver masculino do projeto Visible Human.
Fonte: PROJECT VISIBLE HUMAN (https://www.nlm.nih.gov/research/visible/visible_human.html).

Para a aplicação de renderização, foi utilizado apenas o conjunto de dados que representam a cabeça, disponibilizado pelo projeto Visible Human. A decisão de trabalhar com apenas uma parte do corpo humano teve como objetivo apresentar uma real utilização da aplicação para a visualização de tomografias computadorizadas. Além disso, as limitações da placa gráfica utilizada também contribuíram com a escolha de se lidar com um conjunto reduzido de dados.

2.4 GTK+

GTK+ ou GIMP Toolkit, é um conjunto de ferramentas multi-plataforma utilizado para o desenvolvimento de interfaces gráficas para o usuário. Seu projeto é licenciado

sob a licença GNU LGPL, podendo ser utilizado na construção tanto de *softwares* proprietários quanto de *softwares* livres.

GTK+ é escrito em C, mas foi projetado para suportar diversas linguagens de programação além do C/C++, como por exemplo: Perl, Java, Python, Ruby, PHP, etc.

Originalmente o GIMP Toolkit foi desenvolvido para o sistema X Window, porém o conjunto de ferramentas cresceu ao longo do tempo incluindo suporte para outros sistemas conhecidos. Hoje o GTK+ pode ser utilizado em sistemas Linux, Windows e Mac OS X.

GTK+ oferece uma grande quantidade de recursos para o desenvolvimento de interfaces gráficas, como por exemplo: aparência nativa, suporte a temas, suporte a UTF8, acessibilidade, localização, abordagem orientada a objetos, etc.

O conjunto de ferramentas se tornou bastante popular e atualmente possui uma grande comunidade de desenvolvedores e também mantenedores, que estão presentes no núcleo de empresas como Red Hat, Novell, Lanedo, Codethink, Endless Mobile e Intel.

3 DESENVOLVIMENTO

Durante este trabalho foram feitas modificações na ferramenta de visualização volumétrica com o intuito de realizar melhorias em relação ao desempenho e também em relação a qualidade de visualização do objeto. Foram implementadas as seguintes modificações:

1. Adição de uma janela de opções;
2. Redução da memória utilizada;
3. Remoção de informações de ambiente;
4. Visualização do cérebro e da mandíbula.

3.1 O PROJETO

A aplicação trabalhada neste projeto foi desenvolvida por Camargo (2010) como trabalho de conclusão de curso na Universidade Federal de Alfenas UNIFAL-MG. A aplicação faz uso da operação de Ray Tracing para a realização do processo de visualização volumétrica. O computador utilizado para realizar as modificações possui as mesmas configurações do computador utilizado durante o desenvolvimento da aplicação original, processador Pentium® Core 2 Quad 2.83GHz, 8 gigabytes de memória RAM e placa gráfica NVIDIA GeForce GTX 285 com 1 gigabyte de memória.

Para executar aplicação foi necessário atualizar o código fonte com as novas bibliotecas utilizadas pelo CUDA. Como a aplicação foi desenvolvida em 2010 algumas funções que estavam no código não eram mais utilizadas ou foram substituídas por funções semelhantes.

3.1.1 Janela de Opções

Após realizar as atualizações necessárias para o funcionamento da aplicação,

foi adicionada uma janela de opções com o objetivo de tornar o acesso ao menu da aplicação mais interativo. Como já foi descrito no capítulo anterior, algumas operações eram realizadas clicando com o botão direito do mouse sobre a janela de visualização e depois selecionando a opção desejada. Para simplificar a utilização da aplicação foi adicionada uma janela de opções, inicialmente com as operações de limiarização, filtragem linear, definição do espaçamento entre as amostras utilizadas durante o Ray Tracing e a definição das escalas de cor e de brilho. A interface gráfica foi criada utilizando o conjunto de ferramentas do projeto GTK+.

Para a implementação da janela de opções foi feito o uso da chamada de sistema *fork* para criar e gerenciar um novo processo. Neste caso, foi definido que o processo pai é a janela de opções e o processo filho a janela de visualização. Foi utilizada a chamada de sistema para manter as duas janelas ativas, sem que nenhuma fosse bloqueada enquanto a outra está em execução. A figura 13 mostra a janela de opções implementada, já com todas as operações disponíveis.

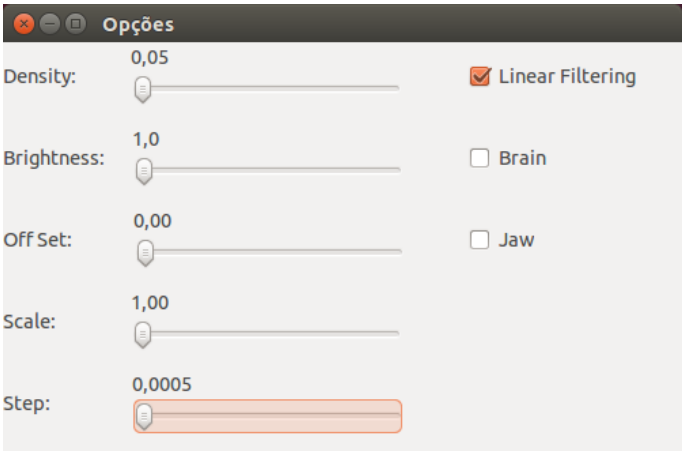

Figura 13 – Janela de opções.

Depois de construída a interface gráfica, foi realizado o compartilhamento de memória das variáveis, para que alterações na janela de opções produzissem modificações no objeto apresentado na janela de visualização.

3.1.2 Estruturas Semelhantes

A partir do momento em que a janela de visualização estava pronta, o passo

seguinte foi entender como a aplicação funcionava e como o algoritmo Ray Tracing percorria o volume de dados e atribuía cor aos elementos da tela.

Após realizar modificações na aplicação, esperava-se ser capaz de renderizar apenas partes específicas da cabeça masculina. A primeira tentativa para atingir tal objetivo foi adaptar o arquivo CUDA responsável pelo Ray Tracing.

Depois de identificada a função responsável por atribuir cor aos pixels na janela de visualização, foram realizados experimentos com o objetivo de isolar algumas estruturas do corpo humano, como por exemplo pele e ossos. O primeiro experimento realizado foi definir um intervalo de cores para serem renderizadas. A ideia principal era que as cores pertencentes àquele intervalo fossem apresentadas de maneira normal e as cores que estivessem fora do intervalo recebessem o valor zero, que representa a cor preta. Com isso esperava-se que apenas estruturas semelhantes fossem renderizadas por apresentar mesmas cores. O resultado deste experimento não foi satisfatório, pois depois de diversas tentativas foi possível notar dois fatos importantes. O primeiro deles foi que a cor de um pixel é influenciada pelos pixels que estão ao seu redor, assim determinando um intervalo muito pequeno de cores o resultado é uma renderização com pouca variação de cor em que não é possível identificar nenhuma estrutura. O segundo fato é que diferentes estruturas podem possuir regiões de mesma cor. Assim mesmo definindo um intervalo, diferentes estruturas foram renderizadas.

O segundo experimento realizado consistiu em modificar o arquivo CUDA. Entretanto, o foco passou a ser a posição em que o elemento é apresentado na janela de visualização. Uma vez que definir um intervalo de cores não trouxe bons resultados, a ideia seguinte foi alterar o código CUDA para delimitar a área a ser renderizada. Ou seja, foram determinados intervalos nos eixos x, y, z com o objetivo de apresentar apenas determinadas estruturas na janela de visualização. Foram obtidos resultados interessantes com este experimento, pois ao se reduzir o volume de dados a ser trabalho foi possível notar uma melhora no desempenho em relação a taxa de quadros por segundo (FPS) na janela de visualização. Além disso, ao se realizar cortes no volume de dados foi possível ver o interior do objeto, sendo possível identificar algumas estruturas. Porém, a finalidade do experimento era isolar estruturas semelhantes e a delimitação da área de renderização apenas realizava cortes na base de dados. Mesmo sendo possível visualizar o interior do objeto, o objetivo de destacar estruturas semelhantes não foi alcançado.

3.1.3 Quantidade de Memória Utilizada

Tendo em vista que alterar o arquivo CUDA não produziu os resultados esperados, a investida seguinte foi realizar modificações no arquivo C++. Porém, ao escolher outro arquivo para ser alterado, a ideia de exibir apenas estruturas semelhantes foi deixada um pouco de lado, o foco passou a ser a quantidade de memória alocada para armazenar o volume e também a qualidade da visualização.

Cada imagem do conjunto de dados possui originalmente uma resolução de 2048 pixels por 1216 pixels, sendo que o volume é formado por um total de 250 imagens. Camargo (2010) definiu que o tamanho do volume utilizado na visualização deveria ser o resultado da multiplicação entre as dimensões da imagem e o valor 320, enquanto a quantidade de memória alocada para a realização de algumas operações deveria ser o valor resultante da multiplicação entre as dimensões da imagem, o valor 320 e o valor 3 representando cada elemento RGB.

Ao executar a aplicação, ficava claro que existia muita informação que não agregava valor a visualização, além disso a quantidade de memória alocada era maior do que a necessária. Pensando nisso, o volume renderizado foi reduzido com o objetivo de diminuir a quantidade de informações de ambiente.

O objetivo definido foi eliminar algumas informações de ambiente que não contribuíam com a visualização. Assim, o tamanho do volume e a quantidade de memória alocada também poderiam ser reduzidos. Ao analisar o volume de dados foi possível chegar à conclusão de que não era necessário renderizar as extremidades das imagens, pois elas não possuem informações sobre o corpo humano. Sendo assim, o tamanho do volume utilizado para a visualização e a quantidade de memória alocada poderiam ser menores. Foram realizadas diversas tentativas de renderização com um volume muito menor, entretanto, os resultados obtidos foram sempre objetos deformados, sendo impossível identificar qualquer estrutura da cabeça. Porém, após realizar alguns testes, a quantidade de memória utilizada e o tamanho do volume foram reduzidos de maneira significativa, sem causar deformações no objeto. Assim, essas duas grandezas passaram a ser definidas pelo valor resultante da multiplicação entre as dimensões das imagens e a quantidade total de imagens.

Antes de serem realizadas as modificações a máquina utilizada não era capaz de executar a aplicação caso algum outro aplicativo que utilize uma grande quantidade

de memória já estivesse sendo executado, como por exemplo o navegador Google Chrome. Após realizar as modificações, a máquina foi capaz de executar a aplicação juntamente com outras aplicações.

Durante a realização das modificações citadas acima foi constatado que Camargo (2010) não utilizou todas as imagens disponíveis durante o carregamento. Então, foram realizados experimentos utilizando todo o conjunto de imagens durante o processo de renderização. Percebeu-se que não utilizar todas as imagens resultava em um objeto formado por cores semelhantes, próximas ao tom de vermelho, dificultando a visualização de algumas estruturas da cabeça devido ao pouco contraste, como mostra a Figura 14. Já ao utilizar todas as imagens durante a renderização, foi gerado um objeto com uma maior diversidade de cores. Permitindo assim, visualizar alguns detalhes que antes não podiam ser percebidos, pois estes passaram a ter cores diferentes, se destacando dos demais, como é possível notar na Figura 15.

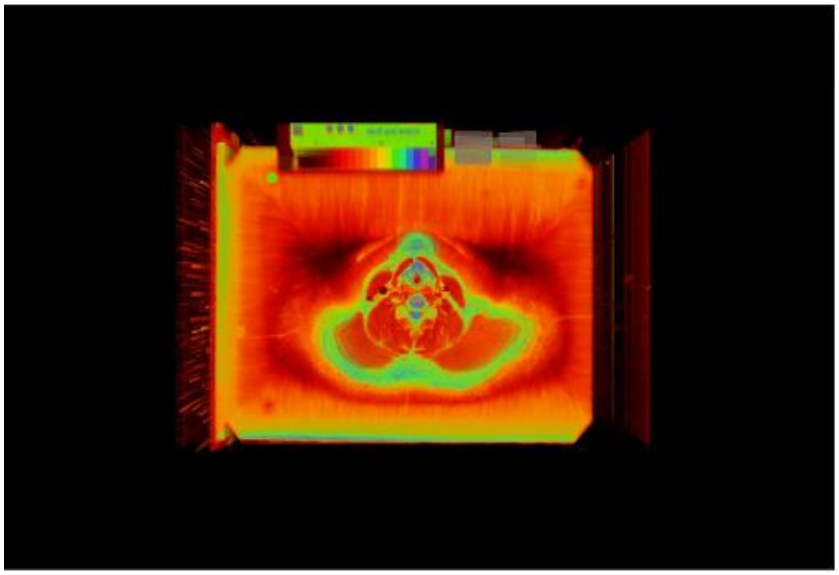

Figura 14 – Renderização do volume de dados.
Fonte: CAMARGO (2010, p. 51).

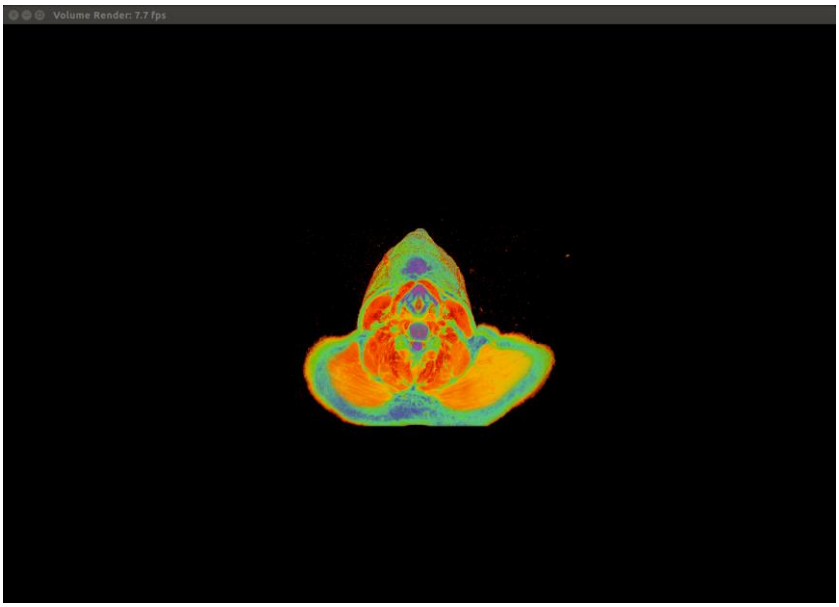

Figura 15 – Renderização após a implementação das modificações.

3.1.4 Informações de Ambiente

A função de corte (*crop*) implementada na primeira versão da aplicação com o objetivo de eliminar informações de ambiente apresentava bons resultados. Entretanto, após a execução da função ainda era possível visualizar muita informação desnecessária, como mostra a Figura 16. Pensando nisso, foram realizadas tentativas para eliminar ainda mais informações que não agregavam valor a visualização. Como não foi possível reduzir ainda mais o tamanho do volume a ser renderizado, foi definido um valor nulo para os elementos que não devessem ser apresentados após a renderização. Nesse caso, foi atribuída a cor preta a esses elementos.

As primeiras informações removidas foram as que estavam nas laterais. Existiam duas maneiras de remover esses elementos. A primeira seria encontrar o intervalo dessa informação na imagem durante o carregamento e atribuir valor zero a ela. A segunda maneira seria modificar o arquivo CUDA para que não executasse cálculos na região onde se encontra essas informações, atribuindo assim a cor preta para a mesma. Após escolher a primeira opção, foram realizadas diversas execuções com diferentes intervalos para determinar a posição da informação a ser removida. Foi possível definir um intervalo que removeu boa parte das informações de ambiente. Entretanto, algumas informações presentes nas laterais e acima da cabeça

permanecerem após as modificações.

Para melhorar os resultados encontrados, foram realizados mais alguns testes. Foi definido então que os elementos abaixo de um determinado valor não contribuíam com a imagem. Assim, ao determinar que as posições no conjunto de dados que possuíssem valores abaixo de 60 receberiam 0, obteve-se um resultado ainda mais significativo, onde foi possível eliminar quase que por completo as informações de ambiente que se encontravam nas laterais da cabeça.

As informações de ambiente acima da cabeça ainda não haviam sido removidas, mesmo com a solução citada acima. Neste contexto, a solução adotada foi remover do volume de dados as imagens que adicionavam essas informações. Assim, foram removidas as cinco primeiras imagens da cabeça, eliminando as informações de ambiente desnecessárias para o contexto da aplicação. Eliminar apenas cinco imagens do processo de renderização não prejudicou a definição das cores da cabeça e nem a qualidade do objeto. A Figura 17 apresenta o resultado após a remoção de quase que todas as informações de ambiente.

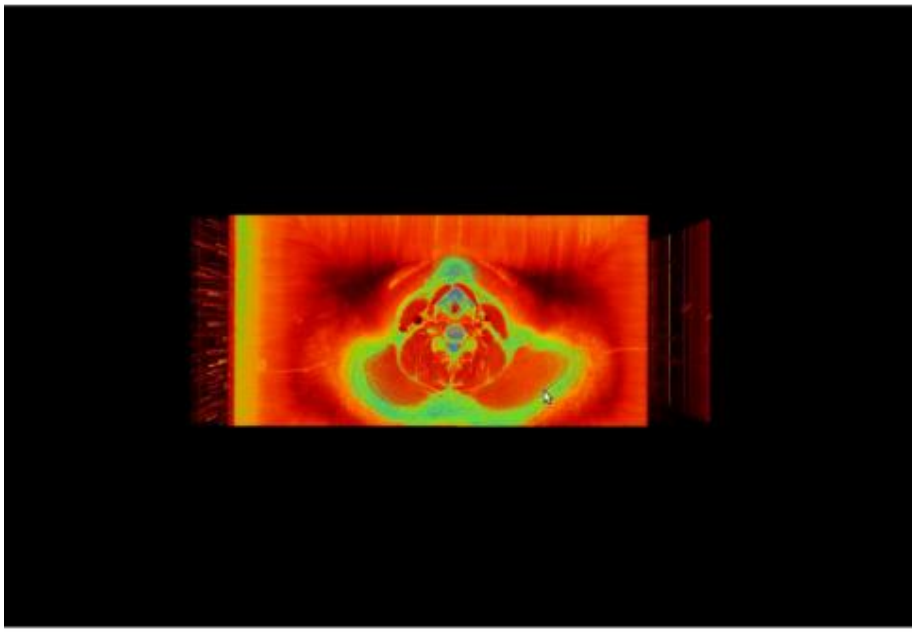

Figura 16 – Utilização da operação de corte (*crop*).
Fonte: CAMARGO (2010, p. 51).

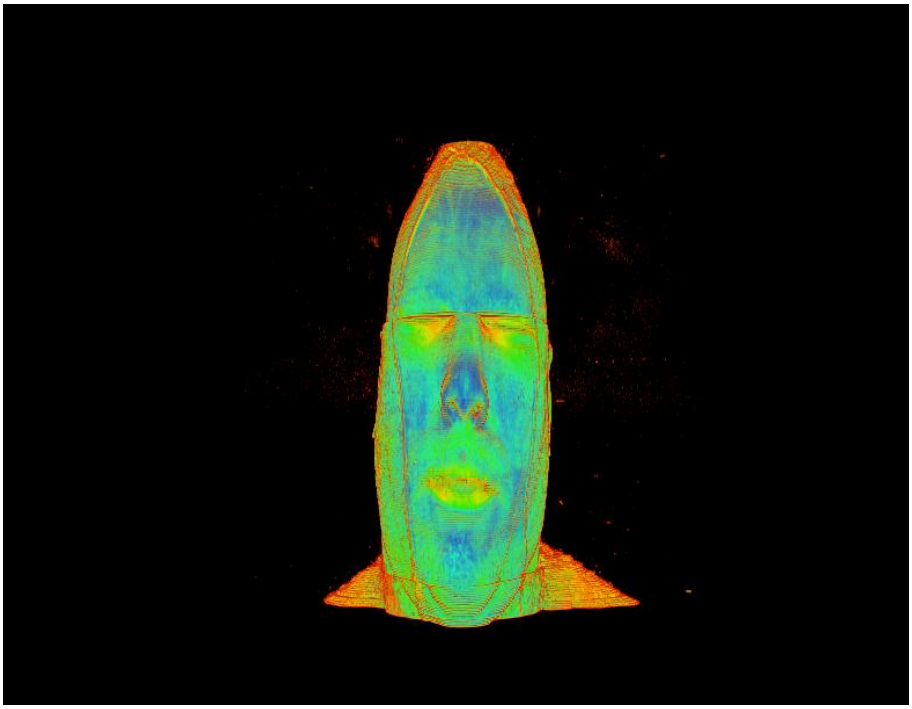

Figura 17 – Remoção das informações de ambiente.

Após essa melhoria na visualização, foram encontradas falhas horizontais próximas ao cérebro, provavelmente causadas durante a captura das imagens, o que prejudicava a qualidade da imagem gerada. Para solucionar esse problema, foi adotada novamente a ideia de remover as imagens que causavam essa deformação. Através de testes de tentativa e erro foi possível determinar o intervalo em que estavam as imagens a serem eliminadas. Após a remoção das imagens o objeto manteve seu sentido naquela região, como se as imagens que compunham aquela área, na verdade estivessem repetidas, fazendo com que aquela região parecesse deformada. Feita a remoção, foi possível visualizar o objeto sem deformações como mostrado na Figura 18.

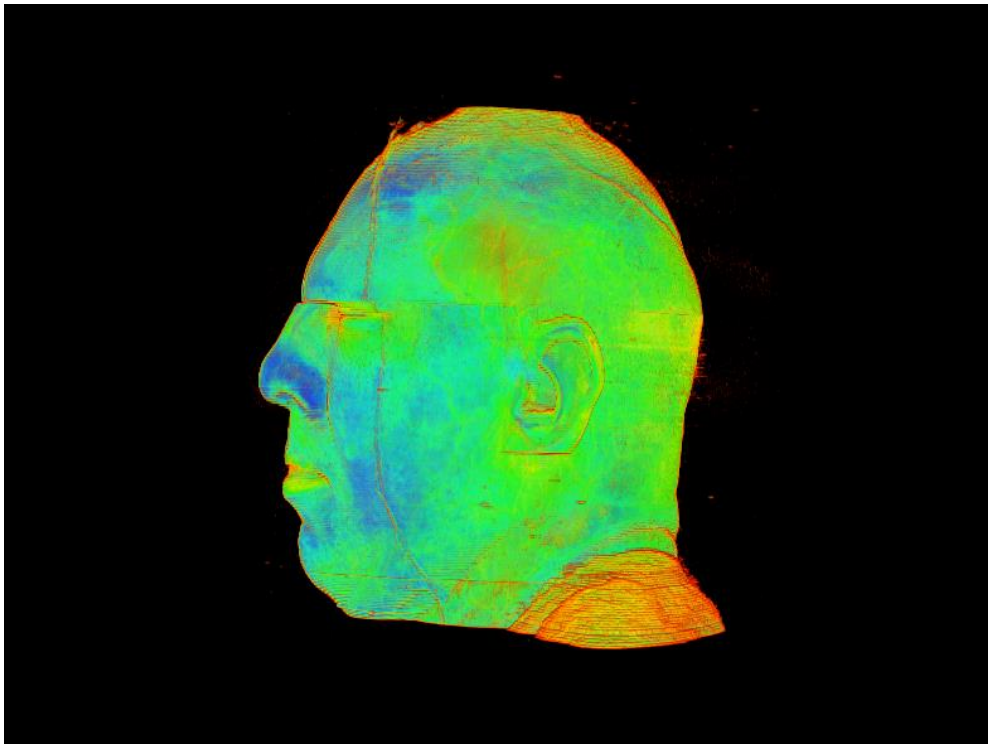

Figura 18 – Renderização após a remoção das imagens.

3.1.5 Visualização do cérebro e da mandíbula

Após remover as informações de ambiente, foram retomadas as tentativas de destacar determinadas estruturas do corpo humano. A função de corte (*crop*) deixou de ser necessária para a aplicação, pois as informações que não contribuíam com a visualização passaram a ser removidas de maneira mais eficiente. Porém, a função foi utilizada como base para a realização dos próximos experimentos.

A primeira modificação feita foi o modo como o conjunto de dados é percorrido. Originalmente foram utilizados quatro laços de repetição aninhados como mostra o Código 1. Os valores escolhidos faziam referência a quantidade de imagens, ao comprimento e largura de cada imagem e pôr fim aos três componentes do pixel. Dessa maneira era mais fácil determinar a posição das informações de ambiente que deveriam ser removidas. Porém, já não é mais necessário percorrer os dados dessa maneira, pois as informações de ambiente já são removidas ao executar a aplicação. Agora a estrutura é simplesmente percorrida utilizando um laço de repetição determinado pelo tamanho do conjunto de dados (resultado da multiplicação entre as

dimensões de cada imagem e a quantidade total de imagens), como mostra o Código 2. Feito isso, a função de corte (*crop*) que atribuía um determinado valor para os elementos dentro de um intervalo e outro valor para os elementos fora desse intervalo foi modificada de maneira a atribuir valores diferentes. Essa alteração no código fez com que algumas áreas da cabeça se destacassem, alterando suas cores e eliminando algumas estruturas.

```
1. unsigned char* crop(size_t size, unsigned char* in) {
2.     int uplim = 1600;
3.     int downlim = 400;
4.     printf("Crop\n");
5.     fflush(stdin);
6.     unsigned char* out = (unsigned char *) malloc(size);
7.     if (out == NULL)
8.        printf("Sem memória");
9.     int ind = 0;
10.    for (int i = 0; i < ns; i++) {
11.        for (unsigned int j = 0; j < nl; j++) {
12.           for (unsigned int k = 0; k < nc; k++) {
13.              for (int p = 0; p < np; p++) {
14.                 ind = i * (nl * nc) + j * (nc) + k;
15.                 if (j < (downlim) || j > (uplim)) {
16.                    out[ind] = 0;
17.                 } else {
18.                    out[ind] = in[ind];
19.                 }
20.              }
21.           }
22.        }
23.    }
24.    return out;
25. }
```

Código 1 – Função de corte (*crop*) na primeira versão do código.

```
1. unsigned char *crop(size_t size, unsigned char* in) {
2.      fflush(stdin);
3.      if (out == NULL)
4.         printf("Sem memória");
5.      for (int i = 0; i < size; i++) {
6.         if(in[i] >= 190 && in[i] <= 230) {
7.            out[i] = in[i] - 100;
8.         }
9.         else {
10.            out[i] = 255;
11.        }
12.     }
13.     return out;
14. }
```

Código 2 – Função de corte (*crop*) após as modificações.

Mesmo algumas áreas estando em destaque, ainda eram necessárias adaptações no código, pois o objetivo era renderizar elementos semelhantes como pele ou ossos e o resultado estava apresentando estruturas diferentes ao mesmo tempo.

Após diversas tentativas, foram deixadas em evidência duas estruturas, o cérebro, apresentado pela Figura 19 e a mandíbula apresentada pela Figura 20. Para alcançar esse resultado foram feitas alterações no código CUDA que consistiram em limitar a área em que são realizados os cálculos que atribuem cores ao objeto.

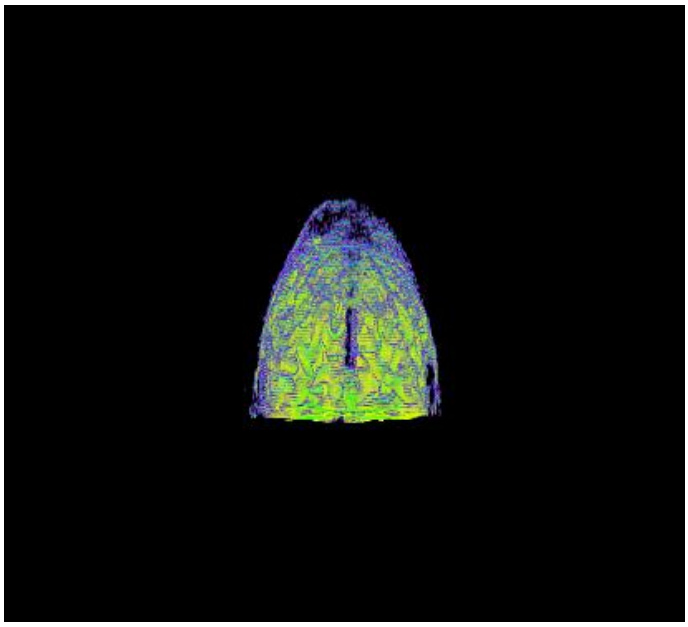

Figura 19 - Renderização apenas do cerébro.

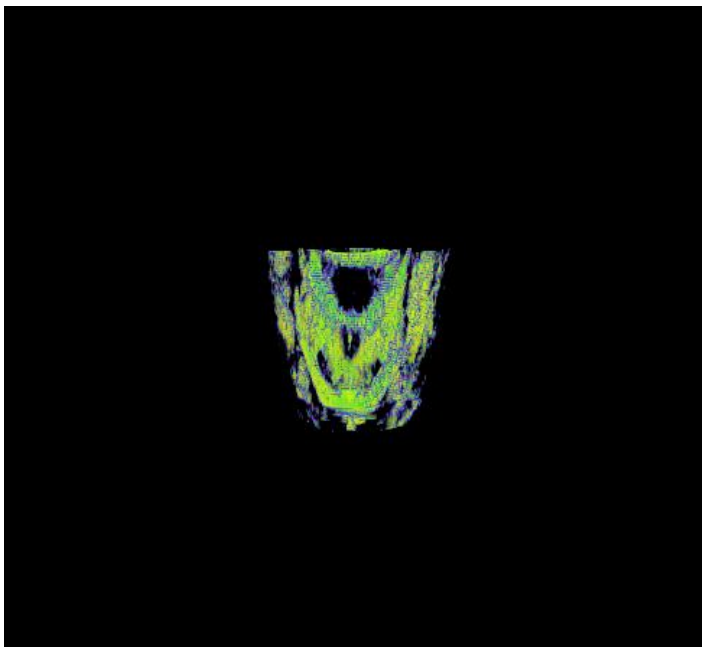

Figura 20 – Renderização apenas da mandíbula.

Ao arquivo C++ foram atribuídas duas funções que determinam as áreas que serão renderizadas de cada estrutura. Já na janela de opções, foram adicionados dois *checkboxs*, um para o cérebro e outro para a mandíbula. Depois que o usuário marca o *checkbox* referente a estrutura que deseja visualizar, os limites da estrutura selecionada são enviados ao arquivo CUDA que ao realizar os cálculos do Ray Tracing atribui zero aos valores fora desse intervalo. Dessa maneira, apenas os dados dentro dos limites são apresentados na tela. O restante da cabeça ainda está presente

na janela de visualização, porém com a cor preta, não sendo possível visualizá-los. As Figuras 21 e 22 apresentam a visualização apenas de estruturas específicas. Entretanto esse objetivo foi alcançado através de modificações na área de renderização e não modificando diretamente o conjunto de dados.

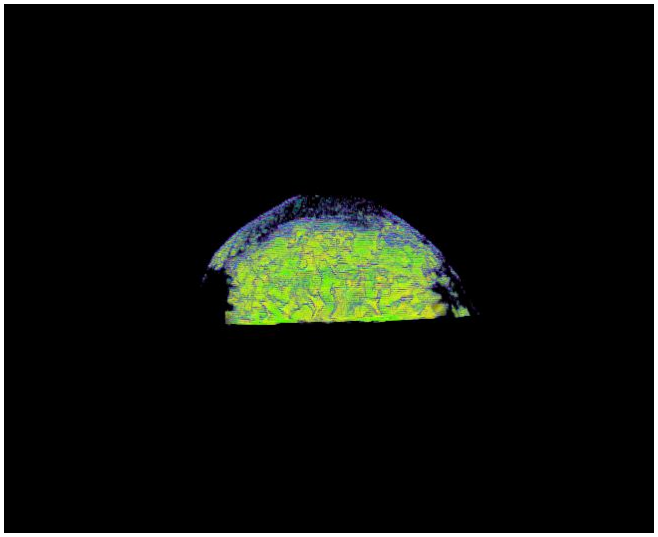

Figura 21 – Renderização apenas do cérebro vista de outro ângulo.

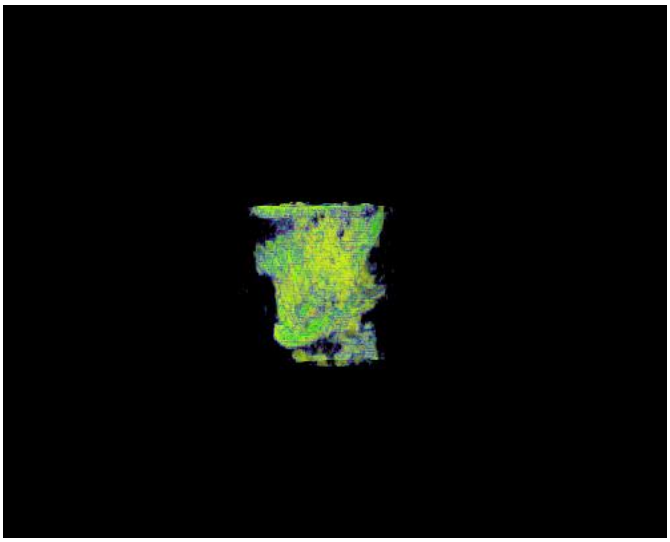

Figura 22 – Renderização apenas da mandíbula vista de outro ângulo.

Para apresentar apenas o cérebro na janela de visualização poderiam ter sido feitas modificações apenas no arquivo C++. Já com a mandíbula não seria foi possível obter o mesmo resultado. Neste contexto, foi escolhido modificar o arquivo CUDA por ser possível renderizar tanto o cérebro quanto a mandíbula e também por questões de desempenho. É possível notar uma melhora em relação a taxa de quadros por

segundo (FPS) ao se limitar a área a ser renderizada, pois foi reduzida de maneira significativa a quantidade de cálculos realizados durante a execução do Ray Tracing tornando a utilização da aplicação mais fluída.

3.1.6 Operações

Mesmo após a realização das atualizações na aplicação, as operações implementadas originalmente continuam funcionais. Por exemplo, através da operação de escala de cor, em que a cor do objeto é alterada para uma frequência mais alta ou mais baixa é possível notar algumas estruturas presentes abaixo da pele como alguns músculos e vasos sanguíneos, Figura 23.

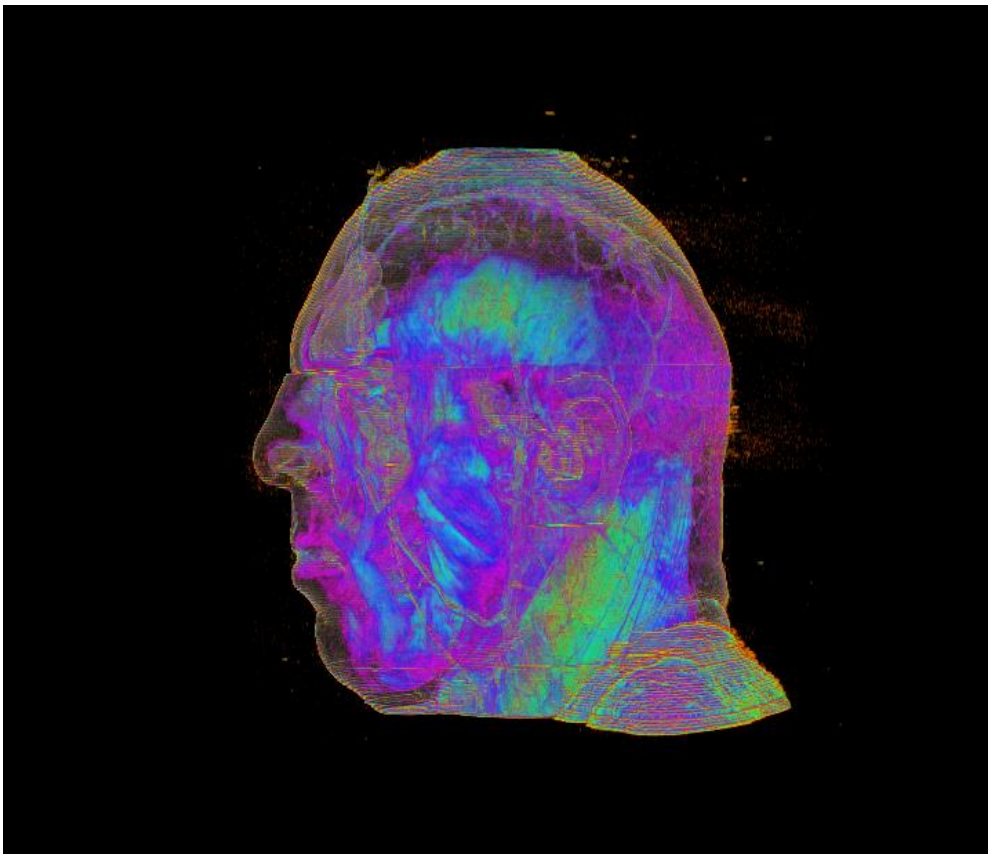

Figura 23 – Renderização com a utilização da operação de escala de cor.

A operação de limiarização implementada com o objetivo de eliminar valores de amostragem dos raios através de um limiar determinado pelo usuário também pode ser utilizada mesmo após as modificações feitas na aplicação. Entretanto, ao utilizá-

la com um limiar baixo o resultado obtido é apenas a mudança de cor do objeto renderizado, Figura 24. Isso ocorre, pois, os elementos que deveriam ser removidos com a utilização da operação com esse limiar baixo, já foram ocultados durante o processo de renderização. Ao executar a aplicação após as atualizações, durante o processo de carregamento das imagens já ocorre uma operação de limiarização para ocultar informações de ambiente. Já ao utilizar valores de limiar mais altos, a operação de limiarização apresenta resultados semelhantes aos resultados obtidos originalmente, com exceção das cores do objeto, como mostram as Figuras 25 e 26.

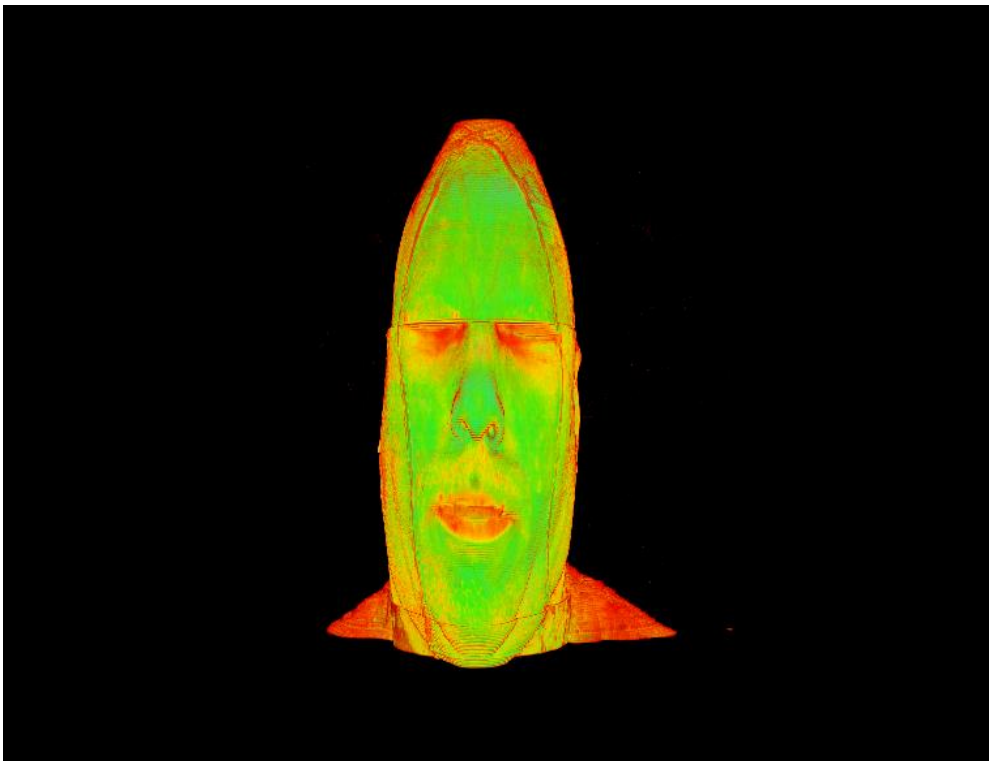

Figura 24 – Renderização utilizando limiarização com limiar igual a 0,15.

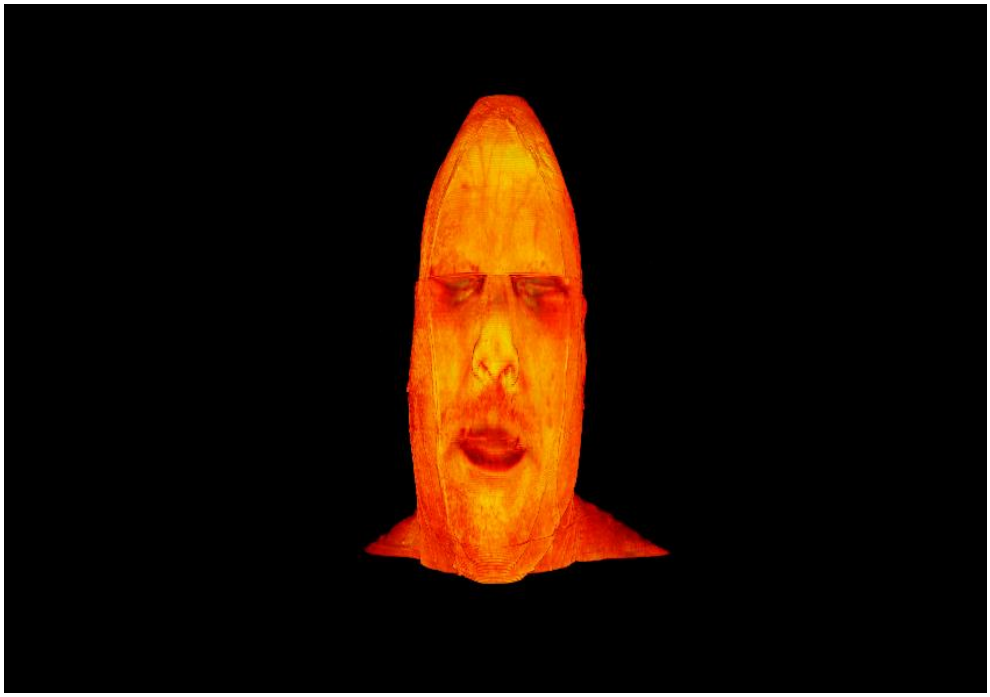

Figura 25 – Renderização utilizando limiariazação com limiar igual a 0,30.

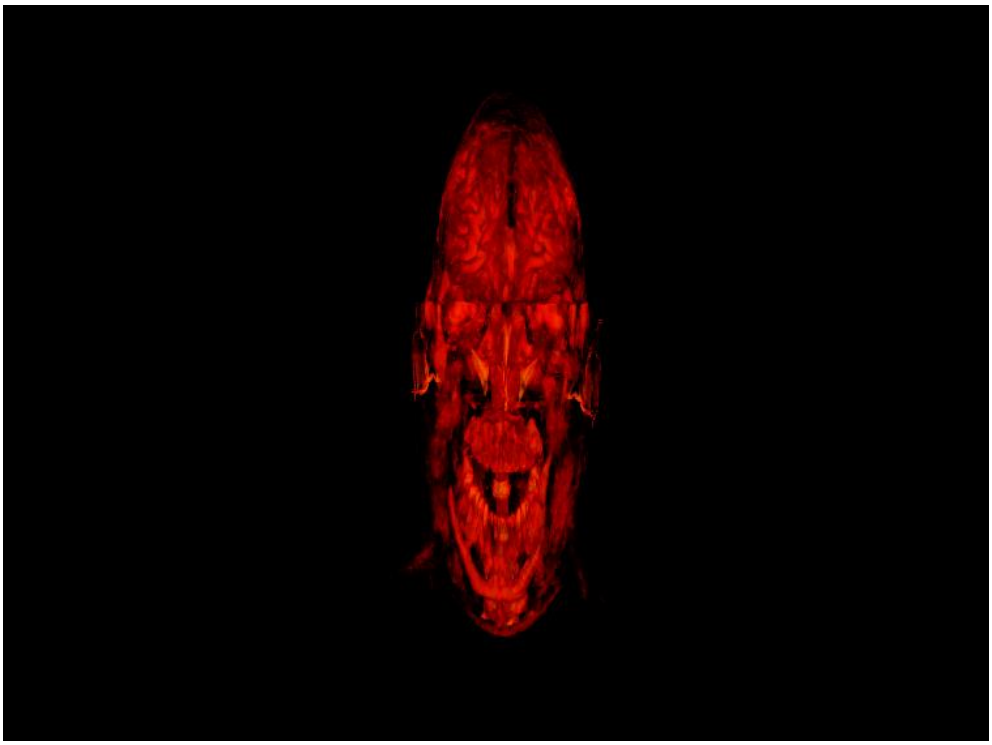

Figura 26 – Renderização utilizando limiarização com limiar igual a 0,65.

Além das operações citadas, é possível modificar a distância entre as amostras utilizadas pelo Ray Tracing. Ao aumentar a distância entre as amostras obtém-se como resultado um objeto com uma qualidade inferior se comparado com uma renderização que utiliza uma distância menor. Entretanto, quanto maior a distância

entre as amostras, menor o custo computacional para a execução do Ray Tracing. Isso ocorre pois quanto menor a distância entre as amostras, maior a quantidade de dados utilizados nos cálculos que determinam a cor do pixel.

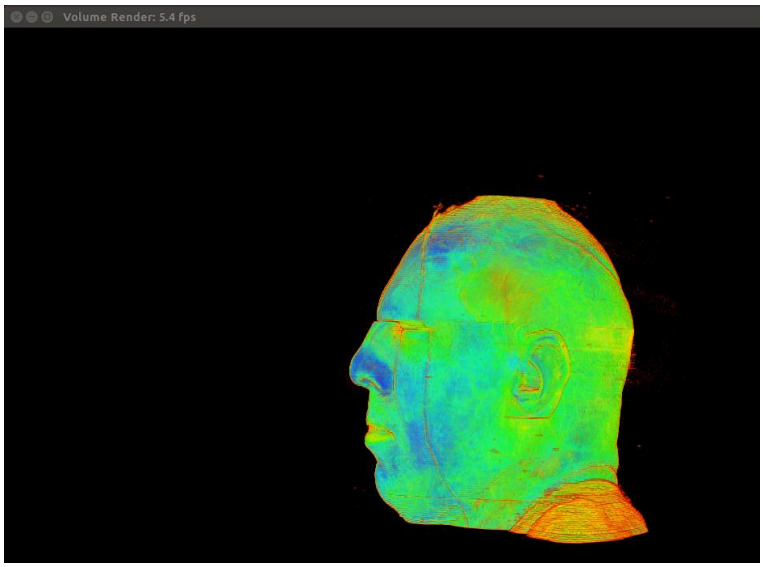

Figura 27 – Renderização com distância entre amostras igual a 0,0005.

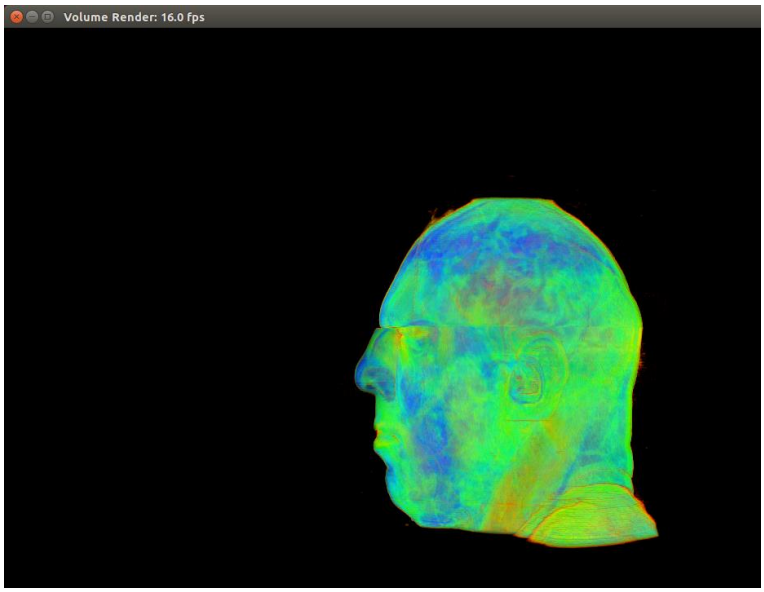

Figura 28 – Renderização com distância entre amostras igual a 0,0020.

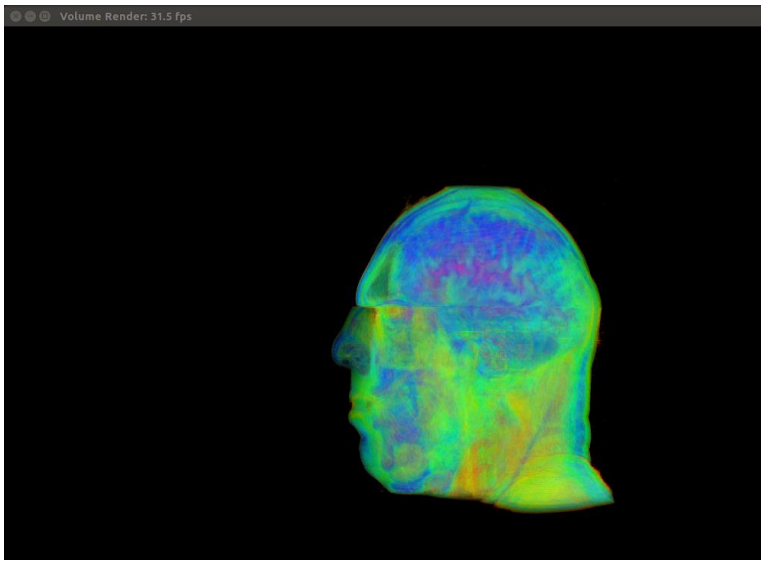

Figura 29 – Renderização com distância entre amostras igual a 0,0060.

É possível visualizar através das Figuras 27, 28 e 29 no canto superior esquerdo que a medida que as distâncias entre as amostras aumenta a taxa de quadros por segundo (FPS) aumenta. Permitindo uma melhor manipulação do objeto, entretanto, com uma menor qualidade de visualização.

3.2 RESULTADOS

Após uma série de modificações foram obtidos resultados positivos que contribuíram com a melhoria da aplicação, tanto em relação ao desempenho, quanto em relação a qualidade da visualização e também em usabilidade.

Para tornar a execução das operações mais simples, foi adicionada a janela de opções. Assim o usuário tem acesso aos recursos da aplicação de maneira muito mais simples. Com a janela de opções o usuário tem disponível uma escala em que ele pode determinar o valor para cada atributo de renderização, como por exemplo brilho ou escala de cor, tornando o manuseio da aplicação mais interessante.

Inicialmente eram alocados 2.390.753.280 bytes para armazenar o conjunto de dados. Feita as alterações, o valor alocado passou a ser 610.140.160 bytes para armazenar o mesmo conjunto de dados. Houve uma redução de aproximadamente 75% na quantidade de memória alocada. Camargo (2010) definiu as dimensões do volume a ser renderizado sendo 2048x1216x320. Após as modificações, as

dimensões foram alteradas para 2048x1216x245, uma redução considerável. O valor 245 é a quantidade de imagens utilizadas e também representa a profundidade do objeto renderizado.

Além disso, houveram avanços em relação a qualidade de visualização do objeto. A renderização realizada após as adaptações do código resultam em um objeto com uma diversidade maior de cores, permitindo assim uma melhor visualização de algumas estruturas. Boa parte das informações de ambiente foram removidas, apresentando apenas dados relevantes para a visualização do objeto. Com isso, a função de corte (*crop*) se tornou desnecessária, sendo utilizada apenas como base para a implementação de novas operações.

Foram adicionadas à aplicação duas novas opções, sendo elas a visualização do cérebro e a visualização da mandíbula. Na janela de opções foram incluídos duas caixas para que o usuário possa selecionar qual estrutura deseja visualizar. As novas opções ocultam informações apresentando apenas o desejado. Desse modo, toda a cabeça ainda está presente na janela de visualização. Porém, a técnica de Ray Tracing não é aplicada naquela região, fazendo com que ela receba valores nulos, que no caso resulta na cor preta, ocultando sua visualização. Como os cálculos da técnica não são aplicados em todo o conjunto de dados, a taxa de quadros por segundo (FPS) é mais alta ao utilizar essas operações. Enquanto essa taxa varia entre 5.0 FPS e 7.0 FPS durante a visualização completa da cabeça, a taxa durante a visualização do cérebro e da mandíbula varia entre 8.0 FPS e 12.0 FPS. Além disso, a taxa de quadros por segundo (FPS) também melhorou durante a renderização completa da cabeça, como pode ser visto na Figura 30.

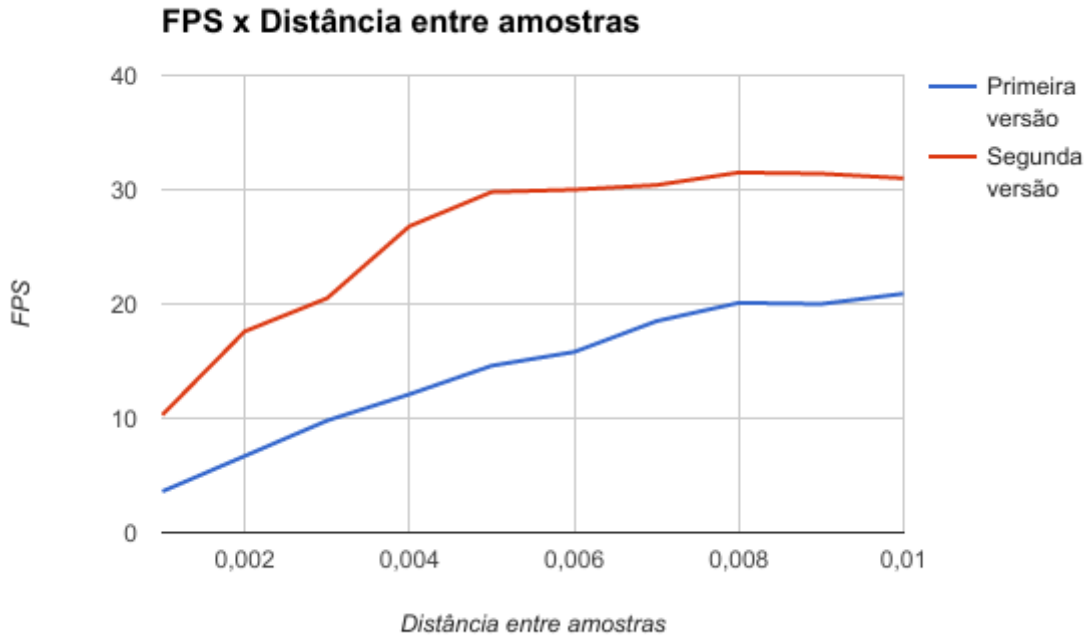

Figura 30 – Comparação entre FPS e distância entre amostras.

É possível perceber ao analisar a figura que a taxa de quadros por segundo (FPS) durante a renderização completa da cabeça após as modificações melhoraram de maneira satisfatória. Na aplicação de Camargo (2010), ao definir a distância entre as amostras com o valor 0,001 o FPS se encontra próximo de 3,6. Já a aplicação modificada com essa mesma distância apresenta um FPS próximo de 10.

4 CONCLUSÕES

Este trabalho objetivou a realização de melhorias na aplicação de visualização volumétrica responsável por utilizar os recursos de placas gráficas programáveis, desenvolvida por Camargo (2010). Foram obtidas melhorias na aplicação em relação a qualidade das imagens visualizadas e em relação a taxa de quadros por segundo. Houve uma melhoria na taxa de quadros por segundo (FPS) da janela de visualização durante a renderização da cabeça completa. Também foi adicionada a janela de opções para auxiliar o usuário na realização de operações sobre a janela de visualização. Além disso, reduziu-se a quantidade de informações de ambiente que não agregavam valor ao objeto. E por fim, adicionou-se as opções de visualização do cérebro e da mandíbula.

No que diz respeito as limitações do projeto, não foi possível reduzir como desejado o tamanho do volume de dados a ser renderizado. Logo, mesmo após a eliminação de informações de ambiente ou após a implementação da visualização apenas do cérebro e da mandíbula, os dados que não são visualizados ainda estão presentes na janela de visualização, estes apenas receberam valores nulos que atribuem a cor preta. Além disso, não foi possível destacar estruturas específicas como pele ou ossos como esperado. Após determinar um intervalo de valores que seriam renderizados acreditava-se que apenas estruturas semelhantes seriam visualizadas. Entretanto, o resultado obtido foi a renderização de diferentes estruturas ao mesmo tempo. Acredita-se que isso ocorreu devido a maneira como o conjunto de dados foi obtido.

Também é importante ressaltar que apesar do progresso alcançado, a aplicação ainda lida com limitações impostas pela tecnologia, como por exemplo o tamanho da memória que a placa gráfica disponibiliza. Com essa limitação torna-se impossível a utilização de conjuntos de dados maiores, como por exemplo a visualização do corpo humano completo. Além disso, a aplicação de Camargo (2010) é uma adaptação do algoritmo exemplo do CUDA, neste contexto é possível que a maneira como é realizado o processo de renderização não seja a mais adequada para o conjunto de dados do projeto Visible Human, prejudicando assim a qualidade da visualização.

Apesar das limitações apresentadas, deve-se levar em consideração que foram

realizadas alterações na aplicação que resultaram em melhorias, principalmente em relação a qualidade de visualização. Além disso, se for utilizada uma placa gráfica que disponibilize uma quantidade maior de memória, será possível realizar a renderização de um conjunto de dados maior. Dessa maneira, pode-se isolar diferentes estruturas do corpo humano utilizando os mesmos métodos aplicados para isolar a cabeça e a mandíbula.

Como trabalhos futuros, serão realizadas execuções da aplicação com o mesmo sistema operacional utilizado no desenvolvimento da primeira versão da aplicação, com o objetivo de verificar se as melhorias obtidas em relação a taxa de quadros por segundo (FPS) se mantêm. Além disso, serão realizadas tentativas de renderizar outras estruturas específicas modificando diretamente o conjunto de dados, não apenas a área renderizada, a fim de proporcionar uma melhor visualização do objeto.

REFERÊNCIAS

MANSSOUR, I. H.; COHEN, M. **Introdução à Computação Gráfica.** SIB, v. 13, n. 2, p. 43-68, 2006.

BRESSAN, P. A. **Visualização Volumétrica Aplicada em Aglomerados de Computadores Convencionais.** 2004. Dissertação (Doutorado em Engenharia Elétrica) – Escola Politécnica, Universidade de São Paulo, São Paulo, 2004.

PAIVA, A. C.; SEIXAS, R. D. B.; GATTASS, M. **Introdução à visualização volumétrica.** 1999. Monografia em Ciência da Computação – Pontifícia Universidade Católica, Rio de Janeiro, 1999.

FARIAS, R.; BENTES, C. **Renderização Volumétrica Paralela de Dados Médicos Irregulares em Clusters de PCs.** In: IV Workshop de Informática aplicada à Saúde, 2004, Paraná. Anais... Paraná: Congresso Brasileiro de Computação, 2004.

CAMARGO, L. T. O. **A Utilização da Tecnologia CUDA em Técnicas de Ray Tracing para Renderizar Imagens Tridimensionais.** 2010. Trabalho de Conclusão de Curso – Universidade Federal de Alfenas, Alfenas, 2010.

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 dez. 2016

MORAES, T. F.; Amorim, P. H. J.; MARTINS, T. A. P. **Visualização Volumétrica de Imagens Médicas através de Raycasting.** In: SEMINÁRIO EM TECNOLOGIA DA INFORMAÇÃO DE BOLSISTAS PCI DO CTI, 3., 2010, Campinas. Anais... Campinas: CTI, 2010. p. 136-144.

IESCHECK, A. L.; SLUTER, C. R.; DEDECEK, R. A. **Visualização volumétrica aplicada às Geociências.** Rev. Pesquisas em Geociências. Vol. 35, n. 1, p. 71-84, jul./ago. 2008.

NOVAES, L.; CAMPOS, C. A.; SILVA, P. M.; CELES, W. **Visualização de Grandes Geobodies em Tempo Interativo com suporte a Filtragram por Componentes e Valores.** In: Workshop of Undergraduate Works, 2012, Ouro Preto. Anais... SIBGRAPI, 2012. p. 144-149.

RÚBIO, C. A. **Estilização e visualização tridimensional de tumores intracranianos em exames de tomografia computadorizada**. 2003. Dissertação (Mestrado em Informática) – Setor de Ciências Exatas, Universidade Federal do Paraná, Paraná, 2003.

MANSSOUR, I. H.; FREITAS, C. M. D. S. **Visualização volumétrica.** Revista de informática teórica e aplicada. Porto Alegre. Vol. 9, n. 2, p.97-126, out. 2002.

CARNEIRO, M. M.; VELHO, L. **Um estudo de algoritmos para visualização simultânea de dados volumétricos e superfícies poligonais.** Relatório Técnico – Pontifícia Universidade Católica, Rio de Janeiro, 2000.

LIMA, C. M. **Uso de renderização volumétrica e realidade virtual para problemas de percolação na engenharia.** Dissertação (Doutorado em Engenharia Elétrica) - Universidade Federal do Rio Grande do Norte, Natal, 2005.

INÁCIO, R. T. **Visualização interativa de volumes com GPU.** Trabalho de Conclusão de Curso - Universidade Federal de Santa Catarina, Florianópolis, 2009.

GOMES, L. A. S. **Recriptografia Assistida por GPU em Servidores de Distribuição de Vídeo.** 2013. Trabalho de Conclusão de Curso - Universidade Federal do Pampa, Bagé, 2013.

GAIOSO, R. R. et al. **Paralelização do algoritmo Floyd-Warshall usando GPU.** In: XIV SIMPÓSIO EM SISTEMAS COMPUTACIONAIS, 2013, Porto de Galinhas, Pernambuco. Anais... WSCAD-SSC, 2013, p. 19-25.

CUDA Parallel Computing Platform. Disponível em: <http://www.nvidia.com.br/object/cuda_home_new_br.html>. Acesso em: 22 nov. 2016.

VASCONCELOS, C. N.; CARVALHO, P. C.; GATTASS, M. **Introdução à Programação de Propósito Geral em Hardware Gráfico.** Revista de Informática Teórica e Aplicada, v. 17, n. 2, p. 270-296, 2010.

The Visible Human Project. Disponível em: <https://www.nlm.nih.gov/research/visible/visible_human.html>. Acesso em: 06 maio 2016.

The GTK+ Project. Disponível em: <https://www.gtk.org/>. Acesso em: 15 jun. 2016.