UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

DIEGO LAZZARI TOMASI

Eletrocirurgia Virtual Baseada em Física

Trabalho de Conclusão apresentado como
requisito parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Prof. Dr. Anderson Maciel
Orientador

Porto Alegre, janeiro de 2013

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Lazzari Tomasi, Diego

Eletrocirurgia Virtual Baseada em Física / Diego Lazzari Tomasi. – Porto Alegre: Graduação em Ciência da Computação da UFRGS, 2013.

49 f.: il.

Trabalho de Conclusão (bacharelado) – Universidade Federal do Rio Grande do Sul. Curso de Bacharelado em Ciência da Computação, Porto Alegre, BR–RS, 2013. Orientador: Anderson Maciel.

1. Eletrocirurgia. 2. Distribuição de calor. 3. Háptico. 4. Phantom. 5. Retorno de força. I. Maciel, Anderson. II. Título.

“We’ve arranged a civilization in which most crucial elements profoundly depend on science and technology.”

— CARL SAGAN

AGRADECIMENTOS

Agradeço ao meu orientador Anderson Maciel, principalmente pela motivação e paciência para a realização deste projeto. Este trabalho tornou-se possível graças aos seus conselhos e ensinamentos.

Pelo apoio e grande ajuda no desenvolvimento do trabalho, agradeço a meus colegas Gustavo Tavares Cabral e Diego Ávila. Também agradeço Lucas Ronchetti, Hernandi Filho e Vitor Jorge, meus colegas de laboratório do grupo de computação gráfica que me ajudaram com ideias e na implementação do projeto.

Agradeço o apoio e encorajamento da minha família, principalmente do meu pai Sérgio Alfredo Tomasi e da minha tia Rosa Luiza Lazzari.

Finalmente, dedico este trabalho a minha amada mãe Juraci Oliva Lazzari, que sempre me encorajou e me deu forças, mas que não resistiu para me ver chegar ao final desta jornada.

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

OpenGL	Open Graphics Library
GLUT	Conjunto de Ferramenta e Utilidades para OpenGL (OpenGL Utility Toolkit).
FPS	Frames por segundo
RGB	Sistema para representação de cores Red, Green, Blue
CG	Computação Gráfica
IDE	Ambiente de Desenvolvimento Integrado (Integrated Development Environment)

LISTA DE SÍMBOLOS

$\delta u/\delta t$	Derivada da função u em relação ao tempo t
u_{xx}	Derivada espacial parcial referente ao eixo x
u_{yy}	Derivada espacial parcial referente ao eixo y
u_{zz}	Derivada espacial parcial referente ao eixo z
α	Difusividade térmica
k	Condutividade térmica
ρ	Densidade da massa
c_p	Capacidade de calor específico
Δ	Divergência do gradiente, operador de Laplace
∇^2	Outra representação para divergência do gradiente, operador de Laplace
γ	Operador que retorna o custo do caminho entre 2 vértices
ϕ	Operador que retorna a temperatura de um vértice
n	Número de vértices
m	Total de elementos da lista de triângulos
r_i	Ponto referente ao vértice i do triângulo
r	Ponto dentro do triângulo
λ_i	Valor da coordenada baricêntrica referente ao vértice i do triângulo
P_b	Ponta do bisturi virtual
P	Qualquer vértice do triângulo mais próximo
V_b	Vetor formado pela subtração de P_b e P
V_n	Normal do triângulo mais próximo
G_o	Posição do god-object
P_h	Posição do Phantom
d_p	Distância entre P_h e o triângulo mais próximo
N	Normal do triângulo mais próximo
F	Força de resistência gerada pelo Phantom

k	Constante de elasticidade
d	Distância entre o ponto real do Phantom e o god-object

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O avanço da tecnologia possibilitou a melhoria e modernização de várias áreas, principalmente a medicina. No entanto, ainda não existem muitas ferramentas novas para auxiliar no planejamento de intervenções médicas, como nas eletrocirurgias. Para resolver esse problema este trabalho objetiva criar um simulador para que o médico possa treinar a cirurgia real previamente através de uma simulação. Foi implementado um protótipo com um modelo baseado em física do processo de distribuição de calor em tecidos biológicos, no contexto de eletrocirurgia virtual. O usuário pode interagir com o sistema através da simulação de um bisturi eletrocirúrgico virtual. A visualização gráfica do modelo se dá usando mapeamento de cor para refletir a distribuição de calor sobre o órgão. A interação com o bisturi virtual é feita pelo dispositivo Phantom Omni, da Sensable. Ele permite manipular objetos virtuais em 3 dimensões com 6 graus de liberdade, e permite fazer rendering háptico com retorno de força. Por fim, o desafio do projeto foi implementar as leis físicas com a maior eficiência possível para que os modelos sejam interativos e executados em tempo real.

Palavras-chave: Eletrocirurgia, distribuição de calor, háptico, Phantom, retorno de força.

Virtual Electrosurgery Based on Physical

ABSTRACT

The advancement of technology has enabled the improvement and modernization of various areas, particularly medicine. However, there are not still many new tools to assist in the planning of medical interventions, as in electrosurgery. To solve this problem this paper aims to create a simulator for the doctor to train the actual surgery previously through a simulation. We have implemented a prototype with a model based on the physical process of heat distribution in biological tissues in the context of virtual electrosurgery. The user can interact with the system by simulating a virtual scalpel electrosurgical. A visualization of the model is given using color mapping to reflect the distribution of heat on the body. The interaction with the virtual scalpel is made by the Phantom Omni device, made by SensAble. It allows you to manipulate virtual objects in 3 dimensions with six degrees of freedom, and allows rendering with haptic force feedback. Finally, the design challenge was to implement the laws of physics as efficiently as possible so that the models are interactive and performed in real time.

Keywords: electrosurgery, heat diffusion, haptics, Phantom, force feedback.

1 INTRODUÇÃO

Com o acelerado desenvolvimento da tecnologia nas últimas décadas, muitas áreas de atuação foram modernizadas e melhoradas devido às contribuições da era digital. Uma dessas áreas que sofreu grandes avanços foi a medicina. A evolução das tecnologias digitais possibilitou uma melhora significativa no diagnóstico e tratamento de doenças e enfermidades, além da descoberta de curas e vacinas. No entanto, as etapas de planejamento de cirurgias de alto e médio risco não acompanharam esta evolução com a mesma proporção, e infelizmente vão pouco além de consultas a imagens médicas tiradas do paciente, como radiografias retiradas de tomografias computadorizadas e ressonâncias magnéticas.

As tecnologias emergentes aplicadas à visualização de dados são de alta demanda e de uso bem sucedido nas aplicações voltadas à medicina atualmente. Nesse contexto, especificamente questões que consideram a visualização tridimensional de características de difícil acesso e compreensão dos pacientes possuem um espaço bastante próspero, como a visualização tridimensional de órgãos internos. Este novo tipo de visualização de dados é mais eficiente em relação à bidimensional, pois pode explicitar detalhes que passariam despercebidos nos planejamentos antigos (MARESCAUX et al., 1998).

Uma modalidade de cirurgia que foi possível graças aos avanços tecnológicos foi a eletrocirurgia. Ela conta com tecnologia consolidada para sua preparação e realização, no entanto, ainda carece de ferramentas para fazer um planejamento da operação de um paciente. Essa tarefa ainda é feita unicamente pela análise de várias radiografias dos órgãos internos do paciente. O problema é que essas imagens nem sempre são nítidas o suficiente, fazendo com o médico deixe passar alguns pequenos detalhes que podem fazer diferença no momento da realização da operação. Um exemplo é a possível existência de pequenos cistos dentro de um órgão, que modifica a região de corte do bisturi, pois os médicos procuram contornar estas saliências.

Com o objetivo, a longo prazo, de minimizar problemas deste tipo, o presente trabalho de conclusão propõe construir a base de uma ferramenta para auxiliar no planejamento de eletrocirurgias, com o intuito de diminuir os erros causados durante as operações. O objetivo é construir um simulador baseado em física realista que simule algumas etapas de uma cirurgia de modo que os processos físicos sejam simulados com muita fidelidade aos processos reais que ocorrem durante a cirurgia. Com este sistema, o médico poderá simular os movimentos que ele mesmo realizará na operação, aumentando as chances de evitar possíveis situações inesperadas durante a cirurgia real.

Como a eletrocirurgia corta os tecidos através do aquecimento dos mesmos, têm-se a necessidade da construção de um modelo de distribuição de calor que seja o mais próximo possível da realidade. Como estamos trabalhando com uma simulação da realidade, são empregadas técnicas já bem difundidas na área da computação e física e também a utilização de heurísticas para buscar a máxima aproximação do nosso universo discreto

ao universo contínuo que é o mundo real.

Para fácil observação do médico/usuário, a visualização é feita usando um mapeamento do calor para cores, que vão de tons de azul a amarelo e vermelho conforme aumenta a temperatura. A interação com o sistema é feita com um dispositivo que se assemelha muito ao bisturi elétrico, apoiando o objetivo de ser o mais fiel possível às cirurgias reais. Por fim, as ferramentas aqui propostas visam buscar eficiência computacional de modo que possam ser executadas em tempo real em computadores pessoais de médio custo, pois também temos a finalidade de democratizar o recurso.

Deve ficar claro que este projeto não pretende construir um simulador completo e operacional, pois isso é um trabalho extremamente complexo e extrapola muito os limites de um trabalho de conclusão. O objetivo é construir um ponto de partida e servir de motivação para trabalhos futuros que continuem este projeto. Mais detalhes sobre alguns itens essenciais que não fazem parte deste trabalho serão discutidos no capítulo 7.

1.1 Trabalhos Relacionados

Existem alguns trabalhos que abordam o problema da eletrocirurgia disponíveis na literatura. Em 2008, surgiu o trabalho que apresentou pela primeira vez um sistema de simulação de procedimentos eletrocirúrgicos com um modelo baseado em física (MACIEL; DE, 2008). Neste trabalho, foi implementado um modelo de distribuição de temperatura nos tecidos biológicos em função do tempo. Além disso, também foi implementada interação háptica através do Phantom Omni, que será mais detalhado no capítulo 5. A principal limitação dessa abordagem é que o modelo de distribuição de temperatura não é baseado nas leis físicas da dinâmica da distribuição de calor.

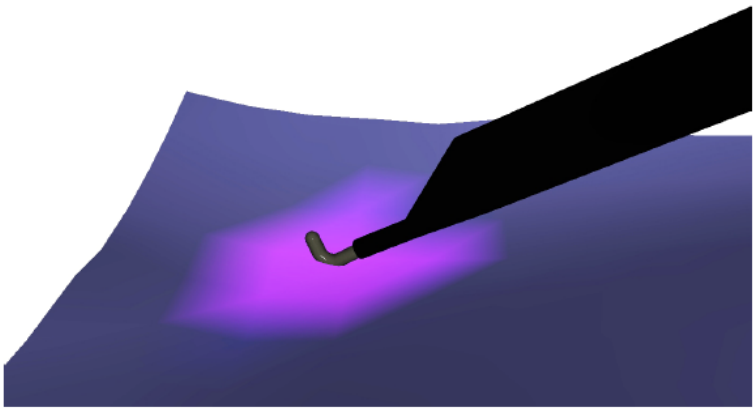

Figura 1.1: Simulação de distribuição de temperatura com interação háptica. Bisturi elétrico está em contato com o tecido gerando calor (MACIEL; DE, 2008).

Outro trabalho vinculado a essa área define um framework para simulações cirúrgicas com interação háptica. O framework segue o modelo model-view-controller (MVC). É uma ferramenta de integração de arquiteturas heterogêneas mantendo o alto desempenho, simplicidade de implementação e extensão. O framework atinge atualizações de mais de 1.000 Hz para interações hápticas e acomoda simulação em rede com atrasos de mais de 1.000 ms, sem perda de desempenho (MACIEL et al., 2010).

Figura 1.2: Na figura da esquerda, o framework de simulação de cirurgia em funcionamento, renderizando os órgãos internos de um paciente. Na figura da direita, está sendo mostrada a interação háptica através do Phantom (MACIEL et al., 2010).

Em 2011 foi publicado um trabalho que pode ser considerado o mais completo desta área. Ele engloba todas as etapas da eletrocirurgia: a fase elétrica, térmica e de deformação. Em todas as etapas foram usados modelos baseados em física, mas não foi implementada interação háptica. Foi feita uma validação do simulador através da comparação dos resultados com experimentos usando fígado de porco, mas com resultados ainda não satisfatórios. Apesar dos esforços dos autores o sistema ainda não está pronto para ser usado pois não consegue ser executado em tempo real, por ter complexidade computacional muito alta. Foi apontado como gargalo do simulador o tempo de cálculo das equações, pois estão sendo usadas operações entre matrizes para resolvê-las. No nosso projeto então, trabalhamos para dissipar esse gargalo usando outro método para calcular essas equações, possibilitando taxas de atualização altas o suficiente para adicionar interação háptica. (KURODA et al., 2011)

Pode-se então dizer que este trabalho tem como inovação a implementação e avaliação de um método discreto para calcular a distribuição de calor em tecidos biológicos, executando em tempo real. Juntamente, foi feita a implementação de uma interação háptica com retorno de força atuando na frequência de 1000Hz. O dispositivo usado foi o Phantom Omni, que fez a simulação do bisturi eletrocirúrgico.

1.2 Organização

Divisão dos capítulos:

- Neste capítulo foi introduzido o problema e seu contexto, solução proposta e metas do trabalho;
- capítulo 2: uma explicação mais detalhada sobre as características de eletrocirurgia para haver um melhor entendimento das decisões tomadas neste trabalho;
- capítulo 3: explicação do modelo de distribuição de calor utilizado e as técnicas, otimizações e adaptações empregadas neste trabalho;

- capítulo 4: discussão e explicação dos pontos mais importantes e significativos da implementação. Descrição dos principais problemas encontrados e das soluções propostas;
- capítulo 5: detalhamento dos testes de desempenho e as conclusões baseadas nos seus resultados;
- capítulo 6: neste último capítulo apresento a conclusão do trabalho, salientando os pontos positivos e negativos e propondo projetos futuros para seguir esta linha de pesquisa.

2 ELETROCIRURGIA

Eletrocirurgia é uma modalidade de cirurgia onde, ao invés de se usar lâminas de corte, é usado um aparelho chamado bisturi eletrocirúrgico (ou caneta eletrocirúrgica). Este bisturi é ligado a um gerador que tem como objetivo transformar corrente elétrica alternada simples em corrente elétrica de alta frequência. Assim, é possível manipular os elétrons, fazendo-os passar através dos tecidos vivos. Uma corrente de elétrons, ao atravessar uma célula, encontra certa resistência. Os íons intracelulares, em resposta à passagem dos elétrons, colidem entre si e contra as organelas intracelulares. Essa colisão produz calor, que em quantidade suficiente, modifica a estrutura do tecido (TRINDADE; GRAZZIOTIN; GRAZZIOTIN, 1998).

Dependendo da regulagem de alguns parâmetros como voltagem, frequência e potência, é possível realizar diferentes tipos de corte com o bisturi elétrico (SILVA, 2004).

- **Corte**: se for usada alta densidade de potência, o aquecimento será rápido e forte indo de 37°C até 100°C. Ocorre explosão da membrana celular, com evaporação do conteúdo intracelular, constituindo o efeito terapêutico de corte;

- **Coagulação**: é obtida com a elevação da temperatura de 45°C a 80°C. Com baixa densidade de potência o aquecimento será lento e fraco, o calor produzido dentro da célula provocará evaporação de água e diminuição do volume celular, constituindo o efeito terapêutico de coagulação.

- **Cauterização** (dissecação): A superfície do tecido seca e forma um coágulo (um fragmento seco de tecido morto), em que os líquidos evaporam até que o tecido esteja completamente seco. Esta técnica pode ser utilizada para o tratamento de nódulos sob a pele, na qual o mínimo de danos para a sua superfície seja desejado. Ocorre com o aumento da temperatura entre 80°C e 100°C.

Existem duas modalidades de eletrocirurgia:

- **Monopolar**, onde o paciente deita em uma placa de metal que faz o papel de eletrodo neutro. O bisturi eletrocirúrgico atua como eletrodo ativo, fazendo com que a corrente corra entre a mesa e o bisturi, passando pelo corpo da pessoa deitada (ver figura 2.1).

- **Bipolar**, essa técnica cirúrgica permite a passagem da corrente elétrica entre os dois eletrodos (ativos e neutro) que normalmente fazem parte de uma mesma pinça. A corrente passa através do tecido apreendido pela pinça e retorna diretamente ao aparelho sem contato com outros tecidos corpóreos. Devido a corrente não passar pelo corpo do paciente, essa modalidade é tida como mais segura.

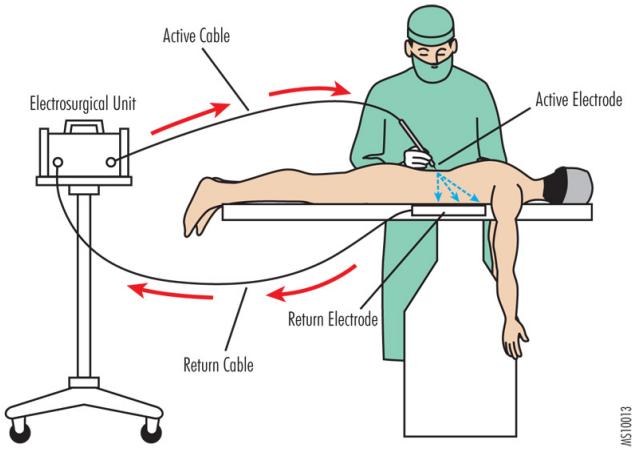

Figura 2.1: Ilustração do fluxo de corrente na eletrocirurgia, saindo do gerador em direção ao bisturi, que esta em contato com o corpo do paciente. Por fim, a corrente volta ao gerador de origem.

Este trabalho não tem o propósito de incluir a simulação da corrente elétrica que faz parte da cirurgia, então não houve muita preocupação com essas modalidades. No nosso caso, pouco importa a técnica empregada, pois ambas concentram a passagem de calor em um único ponto, que é onde o bisturi está em contato com o tecido.

Vantagens da Eletrocirurgia:

Os benefícios da eletrocirurgia incluem a capacidade de fazer cortes precisos com uma perda de sangue limitada, diminuindo os riscos de hemorragia. Também conta com incisão esterilizante, inibição de propagação de germes, rápida coagulação e melhor visualização do campo operatório (BEZERRA, 2009).

Desvantagens da Eletrocirurgia:

Se a potência usada for muito alta, o eletrodo pode causar uma excessiva coagulação ou a formação de zonas necrosadas. Não pode ser usada na presença de determinados agentes anestésicos por serem elementos inflamáveis. Há necessidade de muita atenção durante o emprego do aparelho, porque variações de intensidade na corrente elétrica podem trazer problemas técnicos. Se não for manuseado corretamente, em alguns casos a hemorragia pode ser tardia, gerando complicações na operação (BEZERRA, 2009).

2.1 Planejamento de Cirurgias

Atualmente, não existem boas ferramentas para auxiliar os médicos no planejamento de intervenções cirúrgicas. A maioria delas apenas se baseia na informação visual obtida através de tomografias computadorizadas, ressonâncias magnéticas e raios-x (ver figura 2.2). No entanto, a interpretação destas imagens demanda intenso treinamento do cirurgião e, em alguns casos, não são nítidas ou claras o suficientes para o médico ter uma conclusão com absoluta certeza. Alguns equipamentos de aquisição de imagens mais recentes oferecem ferramentas para a fase de planejamento das intervenções, geralmente contemplando a visualização tridimensional dos volumes adquiridos, a segmentação de estruturas, e o cálculo de volumes. Mas, infelizmente, estes equipamentos são de difícil

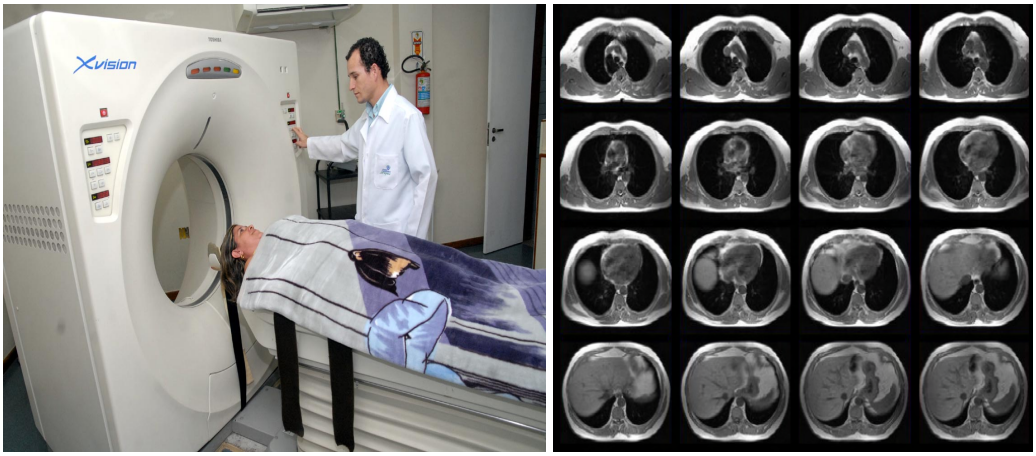

Figura 2.2: À esquerda um paciente sendo colocado dentro de um tomógrafo. À direita exemplos de fatias geradas através da tomografia computadorizada.

acesso e custo muito alto, tornando-se inacessíveis para a maioria dos médicos.

O objetivo primordial deste trabalho é auxiliar no planejamento de eletrocirurgias para fazer com que diminuam os erros médicos. Uma vez que o médico tenha feito um bom planejamento, é provável que diminuam as ocorrências de fatores inesperados durante as operações. Assim, os médicos irão se defrontar com menos frequência com problemas que exijam uma decisão rápida e mais suscetível a erros.

Além de buscarmos que o simulador seja o mais realista possível, também estamos preocupados no preço de custo, pois queremos um produto barato o suficiente para ser executado em computadores populares. As ferramentas que usamos para a construção do simulador permitiram que este objetivo fosse alcançado.

3 MODELO

3.1 Distribuição de Calor

A eletrocirurgia baseia-se em passar calor para os tecidos e produzir um efeito dependendo da quantidade de potência aplicada, a qual interfere na velocidade com que o calor é gerado. Portanto, um modelo de distribuição de calor é essencial e é o primeiro passo para a construção de um simulador de eletrocirurgia.

As equações abaixo descrevem a distribuição de calor (ou variação de temperatura) em uma região uniforme ao longo do tempo:

$$\frac{\delta u}{\delta t} = \alpha \left(\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2} + \frac{\delta^2 u}{\delta z^2} \right)$$

$$\frac{\delta u}{\delta t} = \alpha (u_{xx} + u_{yy} + u_{zz})$$

- $u(x, y, z, t)$ é a temperatura em função do tempo e do espaço;
- $\delta u / \delta t$ é a taxa de variação da temperatura num ponto ao longo do tempo;
- u_{xx}, u_{yy} e u_{zz} são as segundas derivadas espaciais da temperatura em x, y e z, respectivamente;
- $\alpha = k / c_p \rho$ é a difusividade térmica, uma quantidade de material específico dependendo da condutividade térmica k, densidade da massa ρ, e a capacidade de calor específico c_p.

Suponhamos que se tenha uma função u que descreve a temperatura num dado local (x, y, z). Essa função irá mudar ao longo do tempo, conforme o calor se espalha por todo o espaço. A equação do calor é usada para determinar a variação da função u ao longo do tempo e pode ser definida para um número qualquer de dimensões. Neste trabalho é usada a equação para duas dimensões, como será explicado na seção 3.2.

A equação do calor é uma consequência da lei de Fourier de resfriamento. As suas soluções são caracterizadas por um suavização gradual da distribuição inicial da temperatura através do fluxo de calor das áreas mais quentes para as mais frias de um objeto. Em geral, muitos estados e condições iniciais diferentes tendem para o mesmo equilíbrio estável.

A taxa de troca de calor entre dois pontos varia conforme a equação descrita. É interessante salientar que quanto maior for a diferença de temperatura entre dois pontos

adjacentes maior será a taxa de troca de calor. Conforme os dois pontos vão se equiparando em temperatura a taxa de distribuição de calor vai caindo até chegar a zero, quando os dois corpos estarão com a mesma temperatura.

Usando o operador de Laplace, a equação pode ser simplificada e generalizada para equações semelhantes em espaços com número arbitrário de dimensões:

$$\frac{\delta u}{\delta t} = \alpha \nabla^2 u = \alpha \Delta u$$

Onde o operador de Laplace, Δ ou ∇^2 (divergência do gradiente), é tomado em variáveis espaciais. Na matemática, o operador de Laplace ou Laplaciano é um operador diferencial dado pela divergência do gradiente de uma função no espaço euclidiano.

3.2 Operador Laplaciano Discreto

O operador Laplaciano, apesar de ser uma solução correta para o nosso problema da distribuição de calor em um órgão, acaba sendo inviável de ser usado devido ao seu alto custo computacional. Como explicado na seção 1.1, ja foram feitos trabalhos implementando o operador Laplaciano. Para a resolução das equações diferenciais foram usadas operações matriciais, cujo tempo de resposta do escalonamento é muito alto devido ao grande número de operações aritméticas. Como o intuito deste trabalho é implementar um software que resolva a distribuição de calor em tempo real, e sabendo que o computador trabalha em um universo discreto, outro método de implementação precisa ser usado.

O operador de Laplace discreto é análogo ao operador de Laplace contínuo, definido para universos discretos, principalmente como grafos e grids. Esse método tem o objetivo de criar uma aproximação discreta da solução contínua. Ele diz que a soma das diferenças dos valores de um vértice para os seus vizinhos mais próximos é uma aproximação válida equivalente à soma das derivadas segundas parciais. Sendo $\phi : V \to R$ uma função que mapeia um vértice para um valor real e $d(w, v)$ a distância geodésica entre dois vértices w e v no grafo, então o operador Laplaciano discreto Δ agindo sobre ϕ é definido por:

$$(\Delta_\gamma \phi)(v) = \sum_{w:d(w,v)=1} \gamma_{wv}[\phi(w) - \phi(v)]$$

Sendo o nosso modelo tridimensional composto por uma lista de vértices e uma lista de triângulos que define a conexão entre esses vértices, podemos representar o modelo em um grafo. Como sabemos, um grafo é uma estrutura que pode ser representada em duas dimensões, sendo uma matriz de nxn, onde n é o número de vértices. Cada célula contém um campo para definir a vizinhança dos vértices. Em geral contém um número real que define o custo da aresta, sendo infinito ou zero caso os vértices não sejam vizinhos. No nosso caso, estamos definindo o peso das arestas com o valor de um sobre a distância de um vértice até o seu respectivo vizinho. Assim, a quantidade de calor distribuída para um vizinho é inversamente proporcional a sua distância, ou seja, quanto mais longe dois vértices estão, menor será a variação de temperatura.

Outra forma análoga de representar um grafo é através de listas de listas. Temos uma lista principal com todos os vértices do grafo e cada item tem uma lista associada que representa seus vizinhos. Neste trabalho usamos uma estrutura baseada nesta descrição, pois ela se encaixa perfeitamente ao paradigma de orientação a objetos. Desse modo foi mais simples integrar o nosso modelo com as ferramentas escolhidas para o trabalho e também facilitou o desenvolvimento, melhorando sua organização e a inserção de funcionalidades futuramente.

3.3 Etapas da Eletrocirurgia

A eletrocirurgia utiliza corrente elétrica para queimar o tecido e cortá-lo. Com essa descrição podemos dividir o modelo do simulador de eletrocirurgia em 3 etapas principais: elétrica, térmica e estrutural (ver figura 3.1). A etapa elétrica é responsável pela distribuição da corrente e dos potenciais elétricos por toda a superfície. A etapa térmica é definida pela distribuição do calor no órgão. A fase estrutural se refere às consequências geradas a partir da etapa 1 e 2, onde pode ocorrer a deformação ou transformação do tecido.

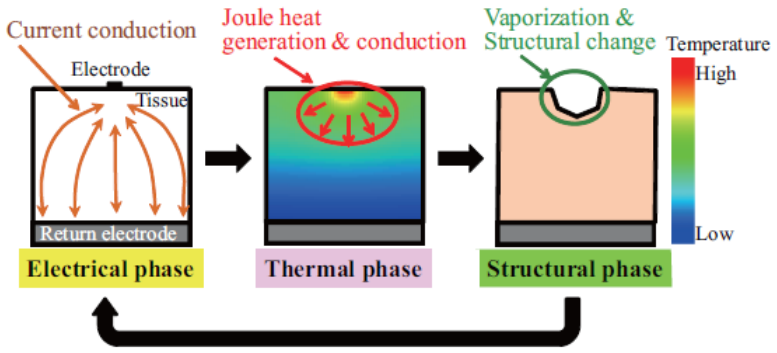

Figura 3.1: Ilustração das três fases principais da eletrocirurgia: elétrica, térmica e estrutural. (KURODA et al., 2011)

Devido à alta complexidade da implementação das três etapas, precisamos escolher apenas uma para ser o foco deste trabalho de conclusão. Foi implementada somente a segunda fase, compreendendo a geração e distribuição de calor no tecido.

4 IMPLEMENTAÇÃO

4.1 Modelos de Órgãos

O trabalho tem como objetivo fazer a simulação de eletrocirurgia em modelos de órgãos 3D. No entanto, não é fácil de se obter um modelo computacional de um órgão ainda dentro do corpo do paciente. Para resolver este problema, contamos com outro software, no qual os participantes deste projeto tiveram profundo envolvimento: SmartContour (DEBARBA, 2009).

A ferramenta SmartContour foi inicialmente desenvolvida por Carlos Dietrich, continuada por Henrique Debarba e por último aprimorada pelo autor deste trabalho, todos junto ao grupo de Computação Gráfica da UFRGS. O software tem como entrada uma série de imagens com fatias do interior do corpo de um paciente, provenientes de tomografia computadorizada ou ressonância magnética. A ferramenta tem como objetivo auxiliar o médico a destacar o órgão pretendido em cada fatia da série, para no final montar um modelo tridimensional usando as informações geradas.

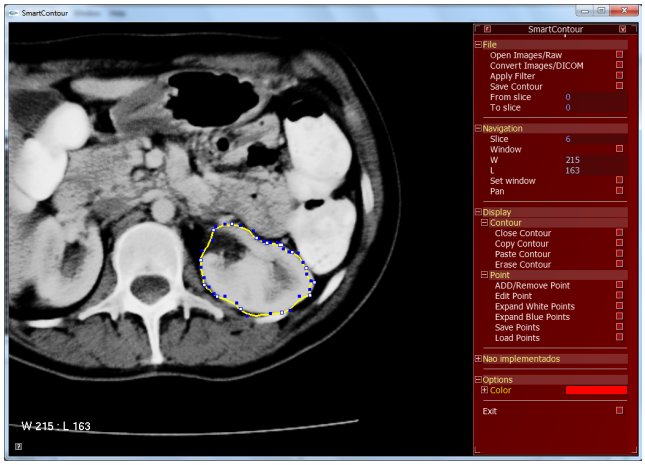

Figura 4.1: Interface do SmartContour. O rim do paciente está sendo marcado nesta fatia

O processo de seleção do órgão em cada imagem pode ser muito trabalhoso e demanda muita paciência da parte do médico. Por isso, o SmartContour facilita este trabalho manual de contornar o órgão em cada uma das dezenas de fatias. Ele usa técnicas para identificar os contornos do órgão automaticamente através de técnicas de processamento de imagens, tornando o processo mais eficiente (ver figura 4.1).

O SmartContour auxilia a criação de modelos tridimensionais de órgãos, que é exatamente o que precisamos para este projeto. O modelo criado é uma malha de triângulos, onde há uma lista de vértices com suas posições no espaço e uma lista com os triângulos

da malha. Com essas informações é possível construir o grafo que precisamos para gerar o nosso modelo de distribuição de calor.

Ocorre que, ao fim do processo, o modelo gerado ainda não é ideal para o nossos fins. Por isso usamos um software de modelagem 3D proprietário chamado 3D Studio Max para refinar a malha e retirar protuberâncias indesejadas (ver figura 4.2). Após passar por esta última etapa, temos o nosso modelo final que é formado principalmente por três vetores:

- **Lista de vértices**: lista da posição x,y,z de todos os vértices que fazem parte do modelo.
- **Lista de triângulos**: lista que compreende todos os triângulos do modelo. Ela é uma lista de inteiros que contém os índices dos vértices que fazem parte de cada triângulo. A cada 3 índices consecutivos temos a definição de um triângulo.
- **Lista de normais**: contém a normal correspondente ao vértice de mesmo índice da lista de vértices.

Figura 4.2: Malha de um fígado gerada a partir do SmartContour e refinada no 3D Studio Max.

4.2 Implementação em Uma Dimensão

O plano de desenvolvimento tem como objetivo solucionar o problema em etapas, tomando o caso mais básico e depois subindo de dificuldade gradualmente. Neste trabalho, o caso mais fundamental é implementar a equação de distribuição de calor usando o método laplaciano discreto em uma única dimensão. Um exemplo de aplicação é a representação do calor se propagando em um fio de cobre. Para esta definição foi encontrado um modelo já previamente calculado como base (HEAT EQUATON EXAMPLE, 2007). Este modelo foi retirado da página sobre este assunto da wikipedia. Foi então construído um protótipo que mostrava a variação de temperatura em um gráfico. O resultado ficou muito semelhante ao modelo de base, mas é claro que acabou sendo uma aproximação. Segue abaixo o algoritmo usado:

$$newHeat = v + (esq - v + dir - v) * dt$$

Nota-se que os vértices são armazenados numa lista e para um vértice de posição n, os itens $n-1$ e $n+1$ são os seus vizinhos à esquerda e à direita respectivamente. Como só há uma dimensão, cada vértice tem exatamente dois vizinhos, com exceção das duas

pontas. Neste caso estamos considerando v como o valor da temperatura do vértice atual, esq como valor do vizinho à esquerda e dir à direita. Usando o Laplaciano discreto, o novo valor de calor é o valor atual mais a soma da diferença entre o vértice e cada vizinho. É necessário incluir um delta t (dt) para que a propagação de calor seja feita em relação ao tempo. Este delta é um valor arbitrário no momento.

O segundo passo foi transpor o modelo para duas dimensões. Neste caso infelizmente não conseguimos encontrar modelos já previamente calculados, o que forçou a avaliação do modelo ser apenas visual. Neste segundo protótipo então, foi necessário adicionar uma resposta visual para o usuário, que se dá no mapeamento de temperatura para cor. Nesta implementação optou-se pela API gráfica multiplataforma OpenGL (Open Graphics Library). A OpenGL, por dispor de recursos com aceleração por hardware, é tida como de alto desempenho. Para estender suas funcionalidades e facilitar o uso, foi usada a biblioteca GLUT (OpenGL Utility Toolkit), cujo principal objetivo é gerenciar tarefas de entrada e saída no nível do sistema operacional, facilitando a tarefa do programador. Como ainda estávamos em fase de testes e experimentações, esse conjunto de ferramentas foi escolhido principalmente por já ter sido amplamente usada pelo desenvolvedor do projeto. Esta decisão acabou acelerando o processo de desenvolvimento e evitando que tempo fosse perdido na aprendizagem de outro ambiente nesta etapa inicial.

Diferentemente da primeira implementação, onde havia apenas uma lista de vértices e cada um tinha exatamente dois vizinhos, em duas dimensões o problema já toma proporções bem maiores. O número de vizinhos é variável e surge a necessidade de criar uma estrutura de dados mais complexa. Um grafo foi escolhido como estrutura para armazenar os dados da malha e poder percorrê-la de forma eficiente.

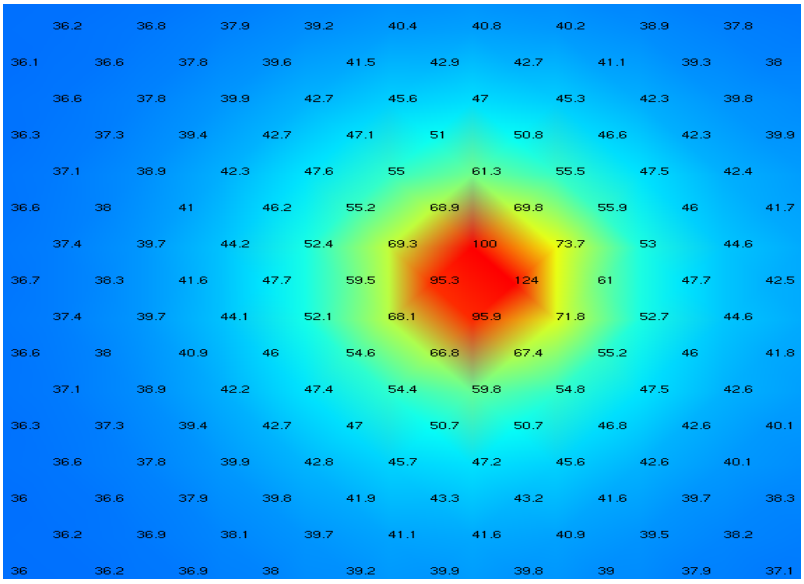

Figura 4.3: Versão em duas dimensões do modelo implementado em OpenGL. Mostra a distribuição de calor a partir de um ponto inicial, com os números desenhados na tela representando a temperatura dos vértices.

Neste protótipo o usuário clica com o mouse e insere calor na malha, que se espalha conforme a equação de distribuição de calor, que será melhor detalhada na seção 4.4.3 (ver figura 4.3). Como explicando anteriormente, essas decisões foram tomadas para auxiliar a fase de testes iniciais do projeto. O próximo passo do plano foi a implementação em três dimensões, que é realmente o que queremos para este trabalho.

4.3 Unity 3D

A Unity 3D é uma engine e IDE de desenvolvimento de jogos. Tem uma poderosa engine de renderização totalmente integrada com um conjunto completo de ferramentas intuitivas para criar conteúdo 3D interativo. O ambiente é de fácil e rápido aprendizado e muito robusto (ver figuras 4.4 e 4.5). Outra característica que influenciou na escolha desta engine foi a facilidade de integração com o Phantom, que será explicado na seção 5.1. Como este projeto tem o intuito de servir como base para um projeto mais completo futuramente, essa ferramenta facilitará o trabalho dos próximos desenvolvedores.

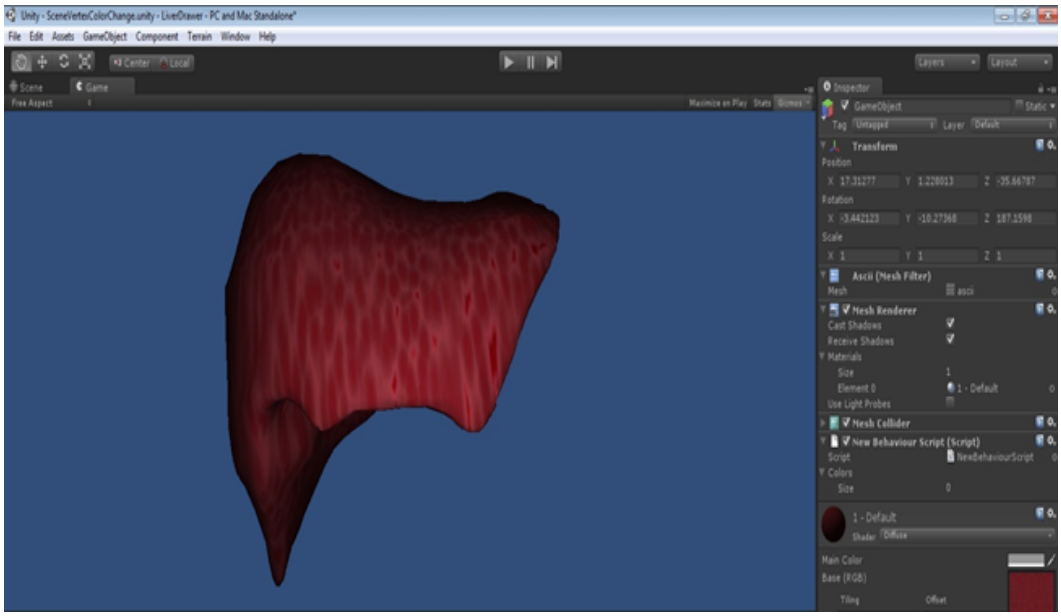

Figura 4.4: Ambiente da Unity sendo usado para manipular uma malha de triângulos que compõe um fígado. Podemos ver que a IDE explicita detalhes importantes da malha na janela da direita, facilitando a sua edição e observação.

Foram usados alguns scripts e métodos fornecidos pela Unity no projeto, assim como a sua física básica, que ajudou em alguns processos.

- **Raycast**: método que traça um raio para calcular se alguma malha foi atingida. Ele retorna várias informações úteis como o ponto do espaço, o triângulo atingido e a distância entre a origem do raio e o ponto atingido.
- **MouseOrbit**: script para movimento da câmera ao redor do órgão.
- **Mesh**: a Unity tem uma classe pré-definida para trabalhar com malhas de triângulos.
- **Vector3**: estrutura para representar um vetor ou ponto de 3 dimensões. Possui um conjunto de métodos e propriedades como operações entre vetores.

Foram usados 4 componentes baseados em estruturas de dados pré-definidas na unity:

- **Câmera**: A câmera principal do simulador, ja vem pré-definida ao criar um novo projeto. Bem intuitiva e de fácil manuseio.

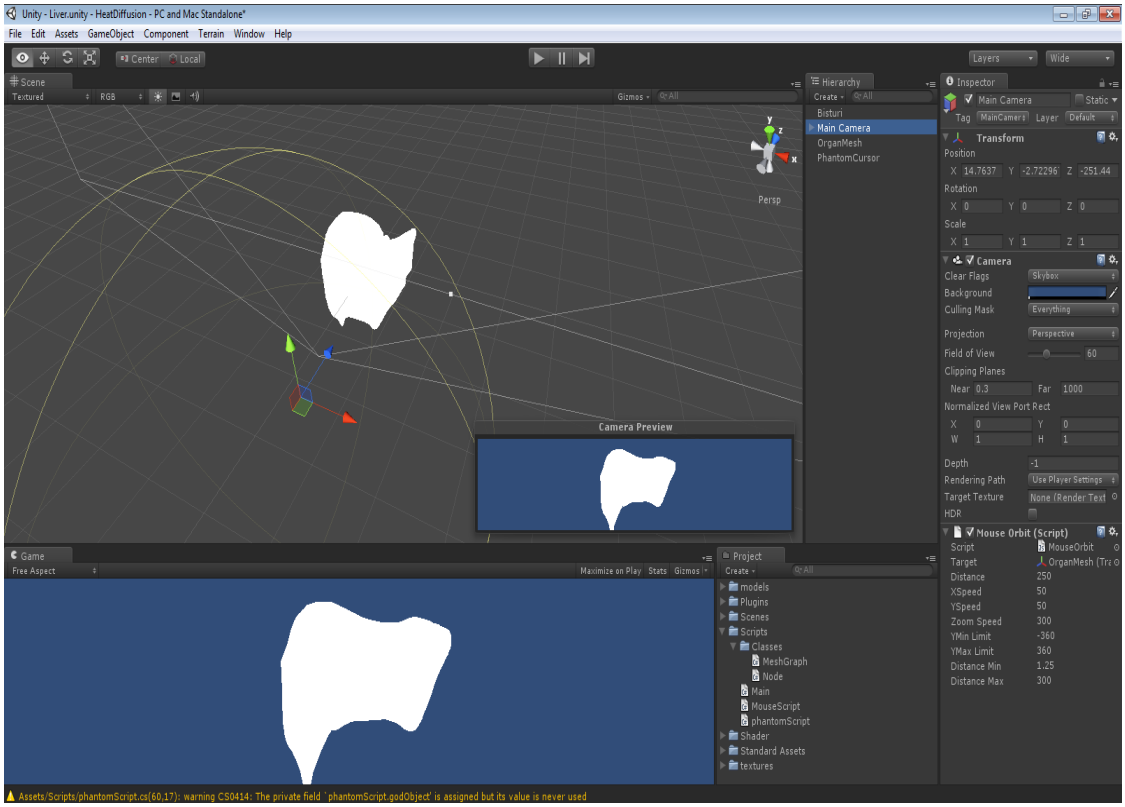

Figura 4.5: Posicionamento da câmera na Unity, sem a necessidade de ser feito através da escrita de linhas de código.

- **Órgão**: do tipo Mesh. É uma malha de triângulos e tem vários métodos e propriedades associadas para facilitar o desenvolvimento.
- **Phantom**: é um objeto de tipo Sphere (esfera). Corresponde a posição real do Phantom no simulador e é invisível aos olhos do usuário. Serve para calcular várias coisas como detecção de colisão e retorno de força.
- **Bisturi**: (Sphere). é a simulação do bisturi virtual. Atualmente, define apenas a ponta do bisturi, mas futuramente deve ser substituído por um objeto que seja baseado no formato e textura de um bisturi eletrocirúrgico.

4.3.1 Shader

Shader é um conjunto de instruções usado principalmente para calcular os efeitos de renderização no hardware gráfico. O maior problema encontrado inicialmente na Unity foi fazer com que os vértices mudassem de cor dinamicamente. Na verdade é bem simples e direto mudar as cores dos vértices, pois o tipo Mesh da unity contém um vetor de cores para cada vértices. Para mudar a cor de um vértice basta atribuir um novo valor ao item de índice correspondente no vetor *colors*. O problema surge na hora de renderizar a malha, pois a maioria dos shaders não leva em conta este campo da classe, levando em conta apenas a cor do material usado na malha.

Existe um shader que ignora a cor do material e desenha a malha com as cores baseadas no vetor colors, mas este shader não habilita o *backface culling* e iluminação. Então, foi necessário modificar este shader para habilitar estas *features* e fazer com que a renderização da cena funcionasse da maneira que esperávamos. Os seguintes comandos no

código foram excluídos do arquivo do shader Particles/Additive.shader:

- **Cull Off**: precisa estar habilitado pois determina quais vértices estão mais próximos do observador, ocultando aqueles que estão atrás.
- **ZWrite Off**: necessário estar habilitado para os cálculos de profundidade no eixo z.

Apesar de muito esforço não conseguimos encontrar um meio de adicionar iluminação no modelo. A linguagem de um shader é bem diferente das outras linguagens de programação e requer um estudo mais aprofundado para adquirir um melhor entendimento. Este problema deverá ser resolvido futuramente.

4.4 Algoritmos

4.4.1 Mapeamento de Cor

Com o devido fim de explicitar visualmente as informações de temperatura para o usuário, a temperatura foi mapeada para cor. Deste modo, conforme a cor vai mudando, o usuário consegue ver o fluxo de calor se distribuindo gradualmente pelos vértices do modelo. O método transforma o valor dado em temperatura para o padrão RGB, sendo R o tom de vermelho, G de verde e B de azul. A mistura destes 3 tons de cores forma uma única cor, que representará a temperatura associada a ela. Cada um dos três bytes de cor RGB tem uma função associada a ele acompanhando a variação de temperatura do vértice.

Os tons de azul representam as temperaturas mais frias, os de amarelo as medianas e os de vermelho as mais quentes (ver figura 4.7). Então temos que a temperatura menores que 0°C são representadas pela cor preta, a temperatura de 25°C é o azul puro (0,0,1)e a temperatura de 100°C e seus valores subsequentes são mapeadas para o vermelho puro (1,0,0).

Figura 4.6: Esta figura tem um de mapeamento de cor parecido com o proposto.

A seguir são mostrados os gráficos de mapeamento de cor em relação a variação de temperatura para cada variável RGB:

Figura 4.7: Mudança gradual de cores conforme aumenta o calor, onde a extremidade esquerda representa 0°C e a extremidade direita 100°C.

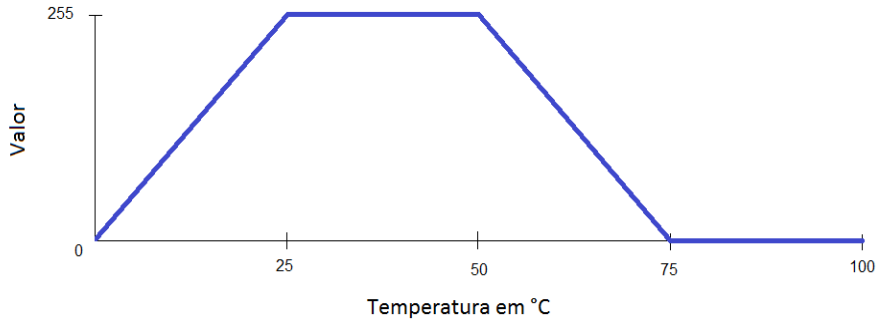

Figura 4.8: Gráfico da cor azul

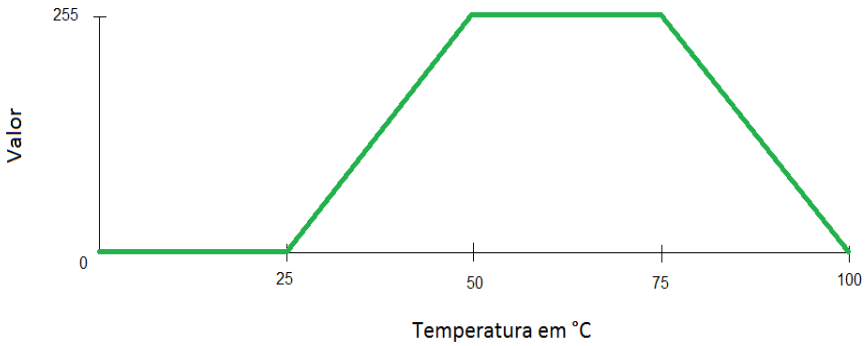

Figura 4.9: Gráfico da cor verde

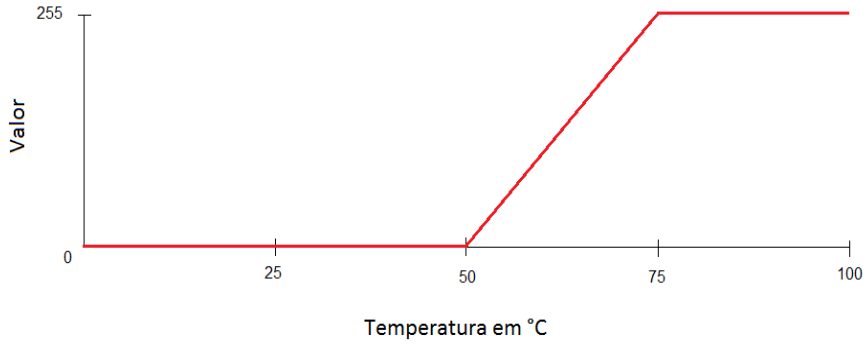

Figura 4.10: Gráfico da cor vermelha

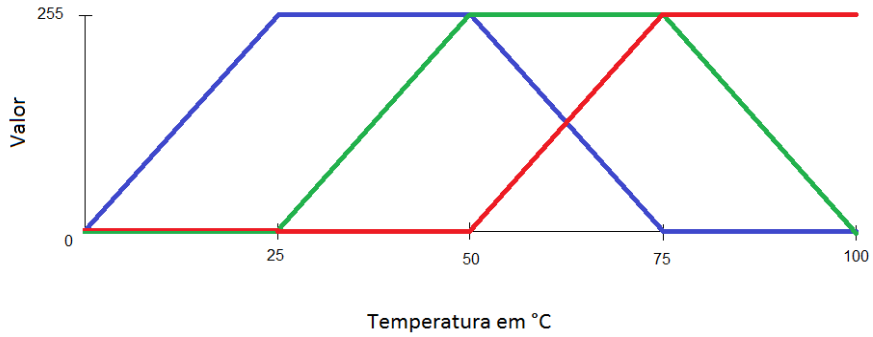

Figura 4.11: União dos gráficos RGB

4.4.2 Grafo de Vizinhança de Vértices

O algoritmo de criação do grafo leva em conta dois vetores: a lista de vértices e a lista de triângulos. A lista de vértices é uma listagem de todos os vértices e suas posições (x,y,z) e a lista de triângulos é uma lista de inteiros que contém os índices dos vértices que fazem parte de cada triângulo. A cada 3 índices dessa lista temos um triângulo. Por exemplo, os índices 0, 1 e 2 da lista formam um triângulo, 3, 4 e 5 formam o seguinte e assim por diante.

A estrutura do grafo então fica deste modo: cada vértice vira um nodo e cada aresta de um triângulo representa uma conectividade. Neste caso estamos considerando vizinhos os vértices que compartilham uma aresta. Podemos construir o grafo usando um algoritmo simples que para cada vértice procura todas as suas ocorrências na lista de triângulos e adiciona suas conexões, com complexidade $O(n * m)$, sendo n o número de vértices e m o número de arestas. É importante perceber que m/3 é igual ao número de triângulos. Apesar de fácil implementação, esse algoritmo vem a ser muito custoso. O tempo para construir o grafo em um modelo de órgão tridimensional com 2500 vértices pode durar mais de uma hora de processamento.

Outro problema encontrado foi que a lista de vértices possui algumas repetições. Isso é feito pelos editores de modelos 3D para auxiliar na renderização e no cálculo de ilumina-ção das arestas. No entanto, esses vértices redundantes devem ser eliminados e unificados em um único nodo no nosso grafo, caso contrário o fluxo de distribuição de calor estaria completamente errado. Então, é necessário mais um passo para a construção do grafo, onde identificamos esses vértices repetidos.

Devido à inviabilidade do algoritmo descrito anteriormente por causa dos altíssimos tempos de resposta, foi criada uma nova solução para substituí-lo. O primeiro passo então é identificar os vértices repetidos e unificá-los em um único vértice. Também devemos corrigir a lista de triângulos para fazer com que os vértices excluídos apontem agora para este novo vértice unificado. Para isso são criadas duas estruturas para agilizar as buscas:

Tabela Hash de Vértices

É criada uma tabela hash onde as chaves são as posições x,y,z e os valores são os índices dos vértices. Desse modo, todos os vértices com mesma posição espacial (redun-dantes) ficarão na mesma posição na tabela. O custo de construção da tabela hash é $O(n)$, sendo n o número de vértices.

Vetor de Posições

É criado um vetor de tamanho igual ao número de vértices. Cada índice corresponde ao seu vértice na lista de vértices. Cada célula contém uma lista de inteiros que corres-

ponde aos índices da lista de triângulos em que o vértice aparece (ver figura 4.12). O custo de construção da estrutura é $O(m)$, sendo m o número de arestas.

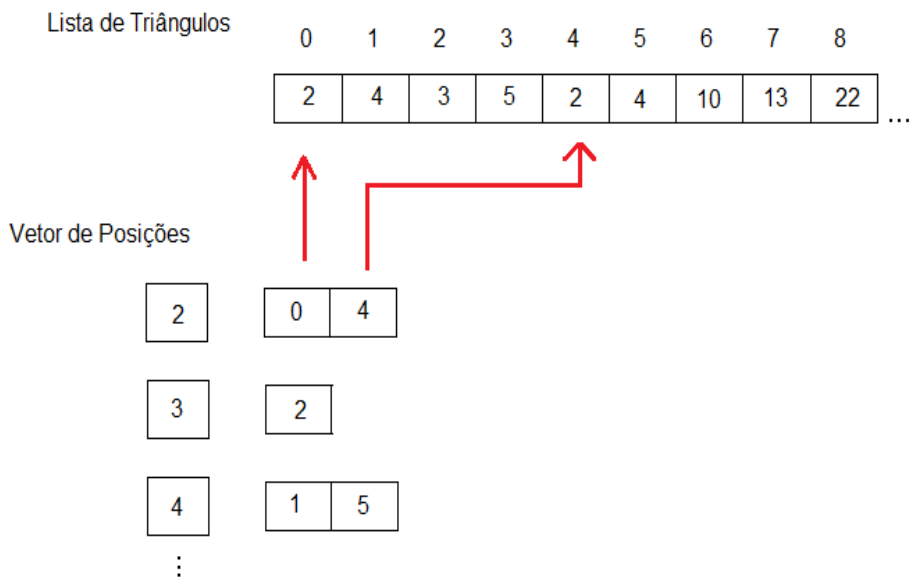

Figura 4.12: Esquema explicando como funciona a estrutura Vetor de Posições. Podemos ver que o vértice de índice 2 está sendo relacionado a uma lista contendo o s índices com suas ocorrências na lista de triângulos (índices 0 e 4).

Eliminar redundância

Cada chave existente na tabela hash conterá uma lista com vértices redundantes de mesma posição no espaço. Queremos que cada vértice que está com a mesma chave torne-se um só com o índice do primeiro vértice. Queremos também atualizar a lista de triângulos para que os vértices excluídos apontem para um mesmo vértice unificado.

Para cada chave da *Tabela Hash de Vértices*, percorremos a lista de índices relativa a esta chave. Elegemos o primeiro índice desta lista (o de menor valor) como sendo o valor do índice unificado. Devemos então eliminar todos os outros vértices da lista. Para cada vértice que será eliminado devemos achar sua posição na lista de triângulos e substituir seu valor pelo seu índice unificado correspondente, explicado no Algoritmo 1. Com a ajuda da nossa estrutura *Vetor de Posições* sabemos quais são suas aparições na lista de triângulos e modificamos diretamente os índices dos vértices redundantes. Em resumo, a lista de triângulos foi percorrida apenas uma vez para atualizar os vértices redundantes.

Criar conexões

Precisamos preencher novamente nossa estrutura *Vetor de Posições*, agora sem os vértices redundantes. Depois, para encontrar os vizinhos de um vértice, basta percorrermos a lista associada a ele na nossa estrutura Vetor de Posições que contém todas suas aparições na lista de triângulos. Os índices junto a ele formam um triângulo. Para achar os índices corretos que dividem uma aresta com este vértice, usamos o resto da divisão de cada uma das suas posições na lista de triângulos por 3 (ver algoritmo 2). A solução é descrita abaixo, supondo o índice atual i:

- resto 0: os vizinhos são i+1 e i+2
- resto 1: os vizinhos são i-1 e i+1

Algorithm 1 Eliminar redundância de vértices

```
for t=0; t<TrianglesList.Count; t++ do
    vertex ← TrianglesList[t]
    Hash[vertex.position] = vertex
    Array[vertex] ← t
end for
for each k in Hash do
    VertexList ← Hash[k]
    vertex ← list.first
    for each i in VertexList do
        TriList ← i
        for each j in TriList do
            newTriangles[j] ← vertex
        end for
    end for
end for
```

- resto 2: os vizinhos são i-1 e i-2

Algorithm 2 Criar Conexões

```
for t=0; t<TrianglesList.Count; t++ do
    Array[TrianglesList[t]] ← t
end for
for each v in VertexList do
    NodesList.Add(new Node(v))
    for each i in Array[v] do
        triangle ← Array[v][i]
        switch triangle%3
        case 0
        NodesList[v].AddNeighbor(TrianglesList[triangle+1])
        NodesList[v].AddNeighbor(TrianglesList[triangle+2])
        case 1
        NodesList[v].AddNeighbor(TrianglesList[triangle-1])
        NodesList[v].AddNeighbor(TrianglesList[triangle+1])
        case 2
        NodesList[v].AddNeighbor(TrianglesList[triangle-2])
        NodesList[v].AddNeighbor(TrianglesList[triangle-1])
    end for
end for
```

Ao todo o algoritmo é equivalente a percorrer 1 vez a lista de vértices e 4 vezes a lista de triângulos, tendo uma complexidade $O(n + m)$. O algoritmo de construção do grafo é executado apenas uma vez para cada modelo e então armazenado, pois mesmo com essa nova solução ainda demora alguns segundos para executá-lo em modelos grandes (milhares de vértices). Após a primeira execução a estrutura do grafo é guardada em um arquivo e é carregada para memória a cada nova execução. Essa operação dura apenas alguns milésimos de segundos.

Estrutura de um nodo do grafo:

- Index: tipo inteiro, Índice do nodo na lista de vértices.
- Heat: tipo real, temperatura atual do nodo.
- newHeat: tipo real, temperatura do nodo após a simulação.
- neighbors: tipo lista de inteiros, lista com índices dos seus vizinhos.

4.4.3 Distribuição de Calor

A implementação da equação de distribuição de calor segue a ideia do operador laplaciano discreto, conforme descrito na seção 3.2. O Algoritmo 3 detalha este procedimento.

Algorithm 3 Distribuição de calor

```
for each Node n in NodesList do
    sum ← 0
    dist ← 0
    for each Node v in n.neighbors do
        dist += Distance(n,v)
        sum += v.heat*dist
    end for
    n.newHeat ← (1-(weight*dist))*n.heat + (sum*weight)
end for
```

A constante *weight*, ou peso, é oriunda da discretização do modelo. O valor desta constante é difícil de definir, quanto mais ela tender a zero, mais deve se aproximar da solução real (contínua). No entanto, ao atribuirmos um valor muito pequeno a essa constante, ocorrerão problemas de arredondamento devido a capacidade de representação do computador. Isso acarretará erros nas operações, o que tornará o modelo mais distante da solução. Então é preciso encontrar um valor intermediário onde o *weight* seja pequeno mas não o suficiente para acarretar erros de representação.

Estamos levando em consideração a distância entre dois vértices para calcular a quantidade de calor que será distribuída entre eles. Isso está sendo feito porque a distância é inversamente proporcional a temperatura de um ponto, ou seja, se um vértice estiver ao lado da nossa fonte de calor, ele estará numa temperatura mais elevada do que um vértice que está mais distante em um dado momento.

O nosso algoritmo considera a perda e o ganho de calor de um nó em relação aos seus vizinhos. No mesmo momento que um vértice está distribuindo calor ele também está recebendo. Essa troca de calor é que faz com que o modelo se estabilize e que finalmente algum tempo após a geração de calor cessar, todos os vértices cheguem a uma mesma temperatura.

O cálculo do calor distribuído é feito baseado na premissa de que um nó passa uma certa parte de seu calor em pequenas porções discretas. A quantidade de calor passada para cada vértice é temperatura atual do nodo distribuidor multiplicada pela distância entre os dois vértices e pela constante peso. Se todos os nodos fossem equidistantes, um nodo passaria a mesma quantidade de calor para cada vizinho a cada iteração. Nota-se que a taxa de distribuição varia para cada iteração, pois a quantidade de calor distribuída leva em conta a quantidade de calor atual do nó, ou seja, quanto maior a temperatura de um nó, mais calor ele vai passar para sua vizinhança em uma iteração.

O cálculo da quantia de calor que um nó recebe de um vizinho é idêntico à descrição anterior. Esse mesmo cálculo é feito para todos os vizinhos e o total de calor recebido é o somatório da temperatura de cada nó multiplicado pela distância multiplicado pela constante peso.

4.4.4 Ponto de Geração de Calor

O ponto de geração de calor no modelo será o ponto onde o bisturi encosta no órgão. No simulador, é um vértice sendo esquentado gradativamente enquanto o bisturi está em contato com ele. Entretanto, o ponto em que o nosso bisturi virtual encosta no nosso modelo virtual pode ser qualquer pixel do órgão e não um vértice do modelo. Como nosso modelo é discreto e só distribui calor entre os vértices, é preciso de uma técnica para distribuir o calor deste ponto qualquer do modelo do órgão para os vértices da nossa estrutura.

A técnica usada é baseada nas coordenadas baricêntricas do triângulo. Deste modo, calculamos as coordenadas baricêntricas do ponto tocado pelo bisturi em relação ao triângulo em que ele esta inserido. Assim, o calor atribuído a este ponto é dissipado entre os três vértices deste triângulo de modo que a quantidade de calor passado é inversamente proporcional à distância.

Sendo $r_1, r_2 e r_3$ os vértices do triângulo, r o ponto dentro do triângulo, a temperatura atribuída a cada vértice leva em conta os pesos λ calculados a partir das coordenadas baricêntricas. A fórmula das coordenadas baricêntricas de r é ilustrado na figura 4.13:

$$\lambda_1 = \frac{(y_2-y_3)(x-x_3)+(x_3-x_2)(y-y_3)}{(y_2-y_3)(x_1-x_3)+(x_3-x_2)(y_1-y_3)}$$

$$\lambda_2 = \frac{(y_3-y_1)(x-x_3)+(x_1-x_3)(y-y_3)}{(y_2-y_3)(x_1-x_3)+(x_3-x_2)(y_1-y_3)}$$

$$\lambda_3 = 1 - \lambda_1 - \lambda_2$$

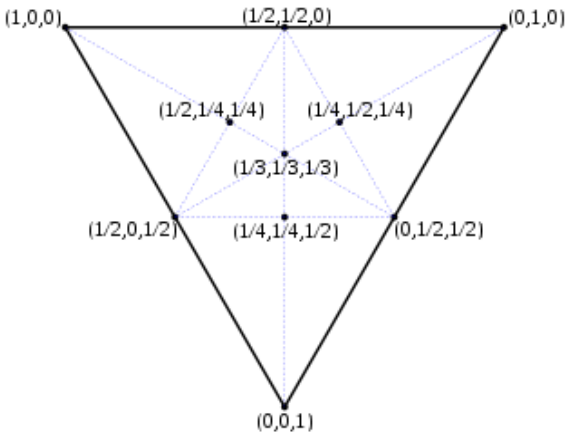

Figura 4.13: Exemplo de coordenadas baricêntricas de um triângulo. Elas são usadas para espalhar o calor para os vértices do triângulo proporcionalmente a distância do ponto de geração de calor.

5 INTERAÇÃO

Além da criação de um modelo de distribuição de calor baseado em física, outro ponto de suma importância para o projeto é a interação. O objetivo é fazer com que o médico (usuário) tenha o mais próximo possível da sensação de estar operando uma cirurgia real. Para isso, estamos usando um dispositivo chamado Phantom Omni, da Sensable, que esteticamente se assemelha muito a um bisturi eletrocirúrgico.

"Uma interface háptica é um dispositivo usado por um ser humano que proporciona as sensações táteis de interagir com objetos virtuais. O dispositivo consiste geralmente de uma ligação electro-mecânica que pode exercer uma força controlável na mão de um ser humano.". (ZILLES, 1995)

O dispositivo háptico é usado para perceber e interagir com objetos virtuais e é muitas vezes combinado com um display gráfico. Renderização háptica é análoga a renderização gráfica com a diferença de que os objetos virtuais são perceptível através do toque em vez de visão. Interfaces táteis foram usados em uma infinidade de aplicações, incluindo treinamento cirúrgico, a investigação do sentido humano de toque, entre outros (STANNEY, 2002).

Para sermos capazes de discutir dispositivos hápticos primeiro precisamos saber algumas coisas sobre o sentido humano de toque. Os seres humanos têm dois sistemas principais para detecção de toque: tátil e cinética. O sistema tátil é constituído pelas extremidades dos nervos na pele. Com estes podemos sentir textura, temperatura e geometria dos objetos. O sistema tátil é sensível a vibrações de até 1000Hz. O sistema cinético é constituído por receptores nos músculos, tendões e articulações. Com estes, podemos sentir as forças sobre nossos membros e saber de qual posição elas vêm (SJÖBERG; YLINENPÄÄ, 2009).

5.1 Phantom

Neste projeto estamos usando o dispositivo háptico Phantom Omni, desenvolvido pela Sensable. Ele permite manipular objetos virtuais em três dimensões com 6 graus de liberdade: andar em 3 dimensões (x,y,z) e rotacionar em torno desses 3 eixos, simulando os movimentos de um bisturi eletrocirúrgico real (ver figura 5.1). Ele permite fazer rendering háptico com retorno de força nos eixos x,y,z. No momento que o bisturi virtual tocar o órgão na tela do computador, o usuário sentirá uma resistência gerada pelo Phantom que impedirá a continuidade do movimento no dispositivo.

A integração do dispositivo com a Unity é bem simples, as funções necessárias são importadas de uma dll criada no grupo de CG da UFRGS que deve ser incluída no projeto. As funções do Phantom usadas no projeto foram:

- Inicialização do dispositivo
- Localização do Phantom nas coordenadas x, y, z
- Botões pressionados
- Retorno de força
- Liberação do dispositivo

Figura 5.1: Phantom Omni

5.2 Detecção de Colisão

O nosso simulador pode possuir várias malhas simultaneamente, que representariam os órgãos internos do paciente. No momento que o bisturi tocar alguma malha deverá ser devidamente tratada a colisão com a mesma. Detecção de colisão sempre é um problema complexo na computação gráfica e no nosso caso não foi diferente. A seguir serão explicados os algoritmos usados para resolver cada problema encontrado pelo caminho.

Tão importante quanto determinar o ponto exato de colisão entre o bisturi virtual e a malha de um órgão é determinar o triângulo mais próximo deste ponto. Encontrar este último no entanto, não é uma atividade trivial. A maneira encontrada para facilitar a busca é encontrar o vértice mais próximo, pois é muito mais simples calcular a menor distância a um ponto do que a um plano (triângulo).

5.2.1 Vértice Mais Próximo

O algoritmo inicia procurando o vértice mais próximo da ponta do bisturi virtual P_b. Estamos supondo que o vértice mais próximo sempre será um dos vértices do triângulo mais próximo. Para encontrar o vértice mais próximo otimizamos a busca percorrendo a nossa estrutura em grafo. Partimos de um vértice e procuramos o seu vizinho mais

próximo do bisturi e repetimos o algoritmo até encontrar o vértice mais próximo. Na primeira iteração partimos de um vértice qualquer do modelo e nas seguintes partimos do último resultado, que provavelmente está perto da nova solução.

1. Calcular distância do vértice atual com P_b
2. Calcular a distância de P_b para cada vizinho desse vértice
3. Se achou vizinho que a distância seja menor torna esse vizinho o vértice atual e pula para 1
4. Se nenhum vizinho é mais próximo então o vértice atual é o mais próximo

Algorithm 4 Vértice mais próximo

```
Let p_0 be the cursor point
index ← -1
while index!=vertex do
  index ← vertex
  dist ← Distance(vertex, p_0)
  for each Node n in Neighbors do
    d ← Distance(n, p_0)
    if d<dist then
      dist=d
      vertex = n
    end if
  end for
end while
```

5.2.2 Triângulo Mais Próximo

Após encontrado o vértice mais próximo de P_b, começaremos a procurar o triângulo mais próximo. Sabemos que este triângulo contém o vértice mais próximo, então o que nos resta a fazer é calcular a distância de P_b até cada triângulo que contém o vértice mais próximo. Queremos o triângulo cuja distância ao ponto seja a menor. O algoritmo foi baseado no trabalho de Maciel e De sobre interação háptica (MACIEL et al., 2008).

1. calcular a projeção do ponto no plano formado pelo triângulo
2. testar se o ponto projetado está dentro do triângulo
3. se verdadeiro retorna distância do ponto ao plano, se falso continua
4. Projetar nas arestas do triângulo
5. retornar a distância do ponto à aresta mais próxima

É importante ressaltar que o triângulo mais próximo é procurado a cada iteração e não somente quando há colisão. Isso é necessário porque usamos o triângulo mais próximo para detectar a colisão do bisturi com a malha, como será explicado a seguir.

5.2.3 Teste de Colisão

A detecção de colisão simplesmente diz se o ponto referente a ponta do bisturi virtual está dentro ou fora de uma malha. Para isso calculamos o produto escalar de dois vetores: V_n e V_b. Tomamos a nossa decisão baseando-se no resultado. Com as informações calculadas anteriormente, agora fica fácil encontrar os vetores que precisamos:

- V_n = Normal do triângulo mais próximo
- $V_b = P_b - P$, onde P pode ser qualquer vértice do triângulo mais próximo

Testamos então o produto escalar entre V_n e V_b e se o resultado for menor ou igual a zero é porque o ponto está fora. Se for maior que zero o ponto está dentro da malha. Simplificando, estamos verificando se os dois vetores estão apontando pro mesmo lado ou para lados divergentes. Se o Phantom estiver dentro da malha, V_b sempre apontará para um direção próxima de V_n, fazendo com que o ângulo entre os dois vetores seja menor que 90°. Mas se o Phantom estiver do lado de fora da malha, o vetor V_b apontará em direção ao triângulo mais próximo, ou seja, na direção oposta de V_n, fazendo com que o ângulo entre eles seja maior que 90°.

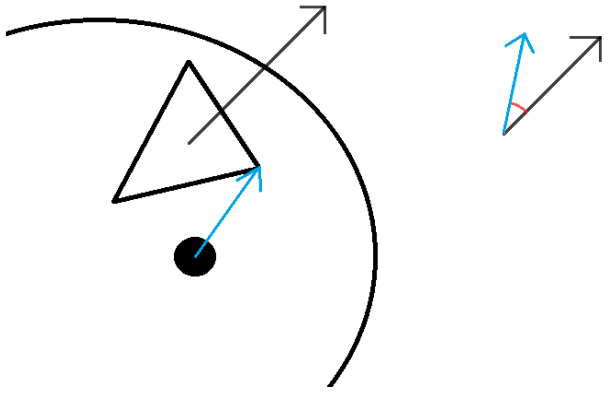

Figura 5.2: Exemplo com Phantom dentro da malha

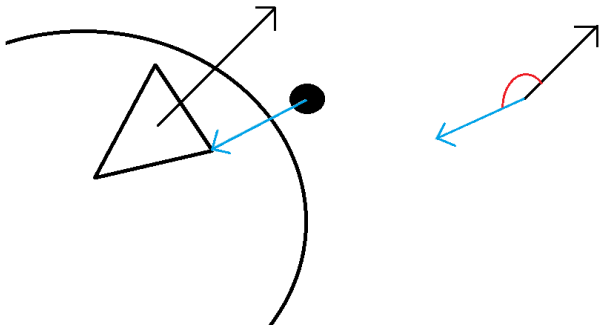

Figura 5.3: Exemplo com Phantom fora da malha

Nas duas figuras temos dois exemplos onde na 5.2 o Phantom está dentro da malha do órgão e na 5.3 está fora. Podemos ver que a seta preta representa o vetor normal V_n e a seta azul representa V_b. Em 5.2 o ângulo formado entre os dois vetores é claramente menor que 90° e seu cosseno é maior que zero. Na figura 5.3 o ângulo está entre 90° e 180°, com um cosseno negativo (ver figura 5.4).

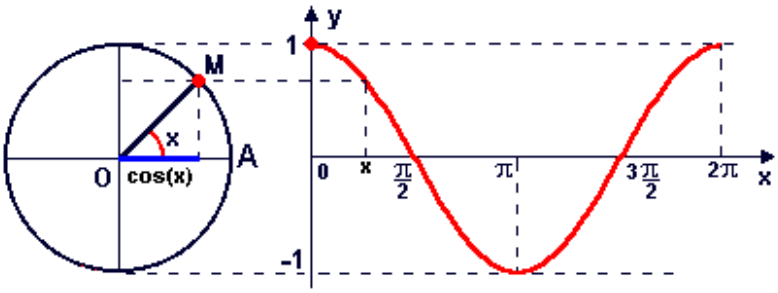

Figura 5.4: Função cosseno

5.3 God-object

Inúmeros problemas surgem ao penetrar um ponto de um objeto háptico em um objeto virtual. Para o devido fim de calcular o retorno de força, não podemos forçar o usuário a parar de penetrar os objetos virtuais com o ponto háptico (Phantom), mas estamos livres para representar a localização virtual da interface háptica. Essa representação visual do bisturi é comumente chamada na literatura de "god-object". Nós temos o total controle sobre o god-object, podemos preveni-lo de penetrar objetos virtuais e podemos fazê-lo obedecer às leis da física do ambiente virtual. (ZILLES, 1995)

Quando não está havendo contato com nenhum órgão, a posição do Phantom e do god-object coincidem. Mas quando há penetração, o god-object para na superfície da malha enquanto o Phantom continua adentrando. Precisamos então calcular a posição do god-object quando o dispositivo háptico estiver dentro de uma malha. Para isso usamos as informações encontradas com o algoritmo de busca do triângulo mais próximo definido em (referenciar minha seção)

- $G_o = P_h + d_p * N$
- P_h = posição do Phantom
- d_p = distância entre P_h e o triângulo mais próximo
- N = Normal do triângulo mais próximo

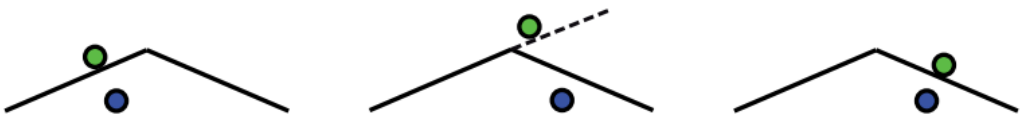

Figura 5.5: Movimento do god-object para acompanhar o movimento do Phantom

5.4 Retorno de Força

Como temos o intuito de simular a interação em um ambiente real, devemos levar em conta as forças que atuam sobre os corpos e as suas reações. No nosso caso, quando o médico toca o bisturi no tecido biológico, este retorna a força aplicada seguindo a terceira

Lei de Newton sobre ação e reação. Temos que então calcular essa força de reação e enviar uma resposta para o usuário. Como estamos usando um dispositivo háptico que simula força, isso poderá ser feito de forma bem realista.

Reunindo todas as técnicas descritas nas seções anteriores deste capítulo, podemos calcular uma penalidade ao penetrar numa malha, que pode ser chamada de retorno de força. Vamos usar a característica do dispositivo háptico para gerar força e criar uma sensação de resistência ao usuário, que aumenta conforme adentra mais fundo na malha. A direção da força será a mesma do vetor normal do triângulo em colisão, ou seja, a força tende a empurrar a mão do usuário para fora da malha (ver figura 5.6).

O cálculo da força é feito da seguinte forma:

$$F = k * d^2 * N$$

- $k =$ constante de elasticidade
- $d =$ distância entre o ponto real do Phantom e o god-object
- $N =$ Normal do triângulo mais próximo

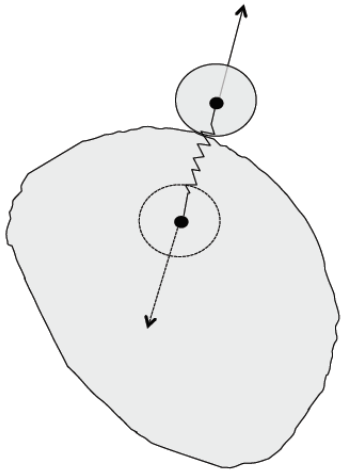

Figura 5.6: Representação da força através de uma mola que une o cursor do Phantom e o god-object

6 TESTES DE PERFORMANCE

Como um dos objetivos do trabalho é buscar eficiência computacional a fim de que o software execute em tempo real, foram realizados alguns testes para verificar o desempenho dos principais algoritmos. Foram usadas diversas máquinas de preços populares para testar modelos de diferentes complexidades.

Foram medidos os tempos de execução dos algoritmos mais custosos, pois estes formam o gargalo de eficiência do simulador. Toda a parte de interação foi desprezada por não causar quase nenhum impacto na medida de tempo de execução. Os cálculos feitas nesta etapa são muito rápidos se comparar com o modelo de fluxo de calor.

6.1 Descrição dos Testes

Nos testes realizados estamos executando a taxa de 250 iterações por segundo, fazendo a atualização da temperatura dos vértices a cada 0.1 segundos. Temos então a taxa de 25 iterações por atualização. A atualização dos algoritmos hápticos é feita na taxa de 1000Hz, ou seja, a cada milésimo de segundo.

1. Execução dos algoritmos de eliminação de vértices redundantes e de criação do grafo com 2 modelos de órgãos diferentes. Os resultados dos tempos de execução medidos em segundos estão na tabela 6.2.

2. Medição do tempo de execução do algoritmo de simulação de difusão de calor em modelos com diferentes números de vértices. Na tabela 6.3 encontramos o tempo que demora para executar 25 iterações do algoritmo.

Para realizar os testes de performance foram usadas malhas geradas a partir dos passos descritos na seção 4.1

6.2 Máquinas Usadas

As configurações das máquinas usadas nos testes estão descritas abaixo. É importante dar atenção aos preços estimados das máquinas, pois foram usados computadores mais populares e de preços mais acessíveis.

6.3 Resultados

Como esperado, os resultados para este método discreto ficaram drasticamente mais rápidos em comparação com os métodos contínuos. Mesmo assim, a partir dos resultados

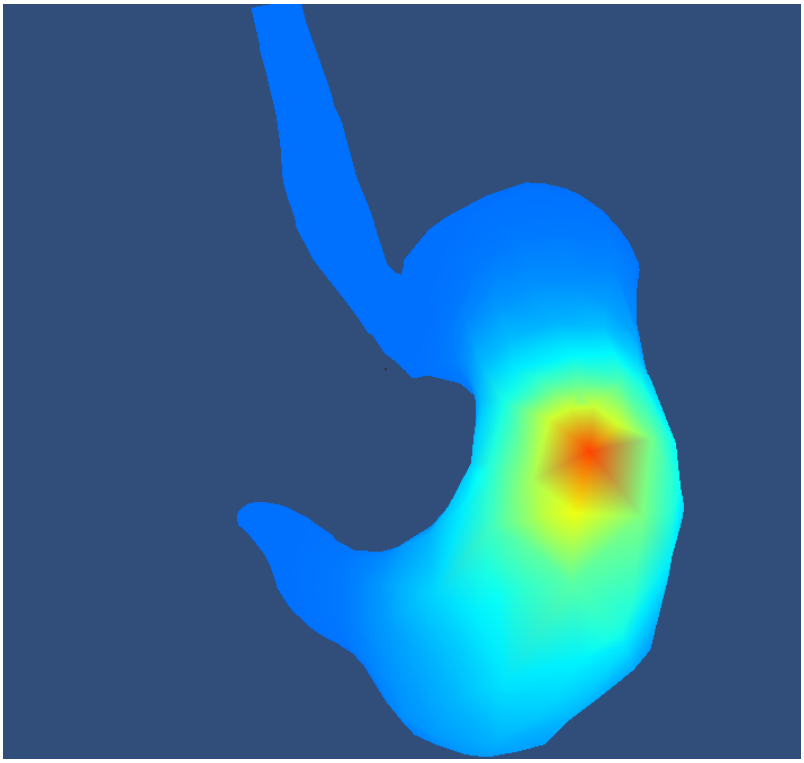

Figura 6.1: Malha de um estômago com 1347 vértices e 1863 triângulos

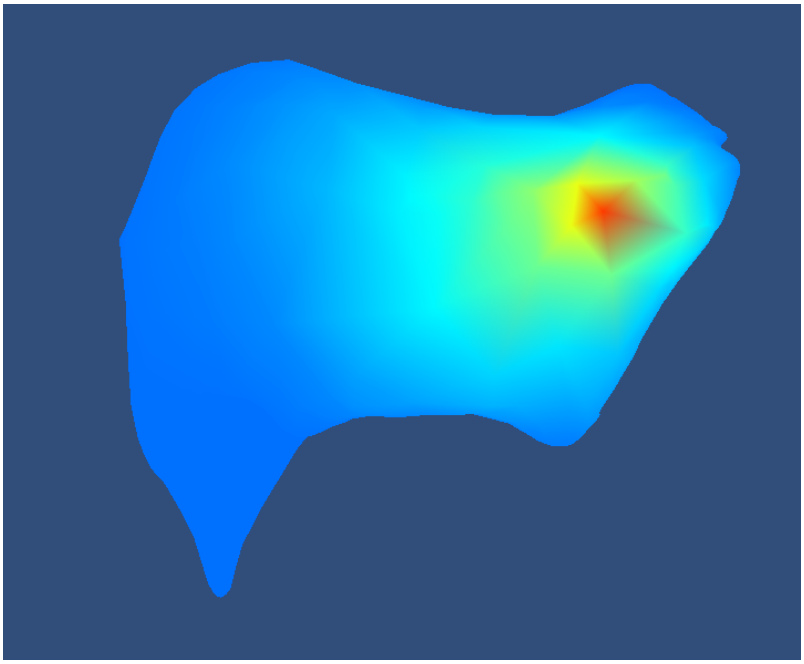

Figura 6.2: Malha de um fígado com 2655 vértices e 3961 triângulos

apresentados ainda não é possível concluir se os tempos são pequenos o suficiente para rodar em tempo real. Apesar de obter boa eficiência numa máquina de preço popular, ainda deve-se levar em conta que faltam muitas outras partes a serem implementadas, como simulação dos potenciais elétricos e deformação da malha. Estas podem vir a demandar muito processamento, o que faria com que a implementação atual se torne defasada.

Uma opção para melhorar o tempo de resposta do sistema seria implementar algumas

Máquinas	Processador	Placa Gráfica	Sistema Operacional	Preço
Pc1	Intel Core2 Quad 2.4GHz	GeForce gts 450	Windows 7 (64-bits)	R$1.200
Pc2	Intel i5 3.0GHz	GeForce gtx 550 ti	Windows 7 (64-bits)	R$1.600
Pc3	Intel i5 2.8GHz	G-Force 7900 GT	Windows 7 (64-bits)	R$1.900

Tabela 6.1: Descrição das máquinas de teste

Vértices	Pc1	Pc2	Pc3
1347	3.5	2.1	3.0
2665	12.5	7.7	20.7

Tabela 6.2: Resultados com construção dos grafos em segundos

Vértices	Pc1	Pc2	Pc3
1347	0.012	0.015	0.014
2665	0.041	0.031	0.035

Tabela 6.3: Resultados simulação em segundos

partes para rodar na GPU, já que o nosso algoritmo de simulação suporta muito bem ser paralelizado. Cada vértice ou nó do grafo pode ser calculado independentemente. Assim sendo, se houver uma placa gráfica com o número de núcleos maior que o número de nós do modelo, todos os vértices poderiam ser calculados em paralelo. Assim o tempo de para calcular a distribuição do calor seria praticamente irrelevante para o resto do sistema.

7 CONCLUSÃO

O modelo de distribuição de calor proposto mostrou ser adequado para o tipo de simulação de eletrocirurgia que o motivou. Mesmo não tendo sido implementado no contexto de um simulador completo, pode-se verificar sua viabilidade para aplicações interativas que exigem acurácia física. Algumas limitações ainda permanecem, principalmente pelo tempo limitado concedido a um trabalho de graduação e porque alguns problemas requerem um estudo mais aprofundado e minucioso. Fica mais fácil fazer uma análise mais completa se dividirmos o trabalho em duas partes: distribuição de calor e interação.

A simulação de distribuição de calor na superfície do órgão teve uma forte base teórica para se afirmar. Esse estudo aprofundado das equações físicas de calor e das suas técnicas para implementação usando o operador Laplaciano discreto surtiu um bom resultado visual. No entanto, ainda existe a falta de uma validação prática. O modelo também sofre com a topologia das malhas trabalhadas, pois elas contém triângulos de formatos e tamanhos muito distintos. A diferença de intensidade de concentração de triângulos em uma região também atrapalha a simulação. Regiões de mesmo tamanho mas com números diferentes de triângulos produzem resultados bem diferentes. Por isso podemos afirmar que este modelo como está implementado agora funciona somente para malhas com triângulos de tamanhos parecidos e com concentração de triângulos uniforme, ou seja, sem áreas muito mais densas do que outras.

A interação feita através do Phantom obteve bons resultados mas também apresenta alguns problemas. A técnica de retorna de força usada ainda não é totalmente ideal para este tipo de trabalho. Como estamos considerando apenas o vetor normal de um triângulo para calcular o vetor de retorno de força, a cada vez que o bisturi passa de um triângulo para outro sofre uma mudança de direção de força às vezes de forma muito brusca. Enquanto isso não seria problema na interação com objetos mais geométricos, pode ser desagradável na interação com formas mais orgânicas, como é o caso de órgãos da anatomia. Esse problema pode ser resolvido de várias maneiras, alguns exemplos de técnicas são interpolação das normais dos triângulos ou a simulação de campos vetoriais.

Por fim, os testes de performance parecem favoráveis e ainda a possibilidade de paralelização do algoritmo é promissora. Apesar de ainda não estar completamente pronto achamos que foi construído um bom ponto de partida para trabalhos futuros nesta linha de pesquisa.

7.1 Trabalhos Futuros

Deformação

Como a eletrocirurgia tem o intuito de cortar o tecido de órgãos internos do paciente,

a malha deve sofrer mudanças equivalentes às sofridas pelo órgão real. Isso implica em adicionar deformação na malha de triângulos. Métodos para deformação adicionam novos vértices e triângulos na malha para criar cavidades na superfície. Isso indica que o número de vértices vai aumentar dinamicamente enquanto a simulação está ocorrendo. A princípio estamos supondo que o número de vértices novos criados pela deformação da malha não deve ser grande o suficiente para que o desempenho do sistema caia significativamente.

Alguns trabalhos sobre deformação de malhas de triângulos aplicados a cirurgia já estão sendo desenvolvidos sob a orientação do prof. Anderson Maciel, orientador deste trabalho. Tudo indica que no futuro esses dois trabalhos se unirão para continuar o projeto do simulador de eletrocirurgia e torná-lo cada vez mais completo e realista.

Distribuição de Potenciais Elétricos

A eletrocirurgia trabalha com a manipulação de corrente elétrica para gerar calor. A quantidade de calor gerada depende do potencial elétrico do órgão no local do toque com o bisturi. Conforme a corrente passa por este ponto os potenciais elétricos da superfície do órgão muda. Conforme o experimento da Gaiola de Faraday, sabemos que o órgão possui campo elétrico nulo em seu interior e que as cargas se distribuem de forma homogênea na parte mais externa da superfície. Por isso é necessário construir um modelo que simule essa alteração nas cargas e no campo elétrico do órgão operado.

Validação

O maior problema desde o início do projeto foi sem dúvida a falta de um modelo já existente e previamente calculado para que pudéssemos tomar como base para os nossos testes. Isso obrigou concentrar a atenção na base teórica e na intuição visual. Com certeza antes de ser testado pelos profissionais da área (médicos) deve ser feita uma validação prática do modelo. No momento não temos recursos suficientes para realizarmos essa tarefa e temos que confiar na nossa pesquisa teórica. Espera-se que no futuro seja dada continuidade ao projeto e que haja mais investimento para melhorar a qualidade do simulador.

Com a existência de um modelo para validação, será necessário ajustar alguns parâmetros do projeto que foram definidos com valores intuitivos. Esses parâmetros compreendem alguns tópicos fundamentais do trabalho como a definição de número de iterações por segundo e o tamanho do delta t, explicados em 4.4.3. Deverão ser feitos testes para encontrar os valores ideais para esses parâmetros para que se assemelhe ao modelo de validação.

Alguns outros parâmetros também foram definidos arbitrariamente, como a taxa de calor gerada pelo bisturi e a taxa de resfriamento do tecido. Infelizmente tiveram que ficar fora do escopo deste trabalho pela complexidade envolvida, pois necessitam de um estudo a parte sobre seus comportamentos.

REFERÊNCIAS

BEZERRA, M. **Noções de Eletrocirurgia e Hemostasia**. Publicado no site, http://pt.shvoong.com/medicine-and-health/1871236-nocoes-eletrocirurgia-hemostasia/.

DEBARBA, H. G. **Ferramentas Para a Segmentação de Imagens Médicas e Segmentação Funcional do Fígado Aplicadas ao Planejamento de Hepatectomias**. 2009.

HEAT Equaton Example. Exemplo de propagação de calor em uma dimensão, http://en.wikipedia.org/wiki/File:Heatequation_exampleB.gif.

KURODA, Y. et al. Electrical-thermal-structural coupling simulation for electrosurgery simulators. In: ENGINEERING IN MEDICINE AND BIOLOGY SOCIETY, EMBC, 2011 ANNUAL INTERNATIONAL CONFERENCE OF THE IEEE. **Anais...** [S.l.: s.n.], 2011. p.322–325.

MACIEL, A.; DE, S. Physics-based Real Time Laparoscopic Electrosurgery Simulation. In: MMVR16: MEDICINE MEETS VIRTUAL REALITY 16. **Anais...** IOS Press, 2008. p.272–274. (Studies in Health Technology and Informatics, v.132).

MACIEL, A. et al. The WHaT: a wireless haptic texture sensor. In: HAPTICS '08: PROCEEDINGS OF THE 16TH SYMPOSIUM ON HAPTIC INTERFACES FOR VIRTUAL ENVIRONMENT AND TELEOPERATOR SYSTEMS (HAPTICS'08), Reno, NV, USA. **Anais...** IEEE Computer Society, 2008. p.217–223.

MACIEL, A. et al. Surgical model-view-controller simulation software framework for local and collaborative applications. **International Journal of Computer Assisted Radiology and Surgery**, [S.l.], p.1–15, 2010. 10.1007/s11548-010-0527-3.

MARESCAUX, J. et al. Virtual reality applied to hepatic surgery simulation: the next revolution. **Annals of Surgery**, [S.l.], v.228, n.5, p.627, 1998.

SILVA, G. A. da. **Eletrocirurgia**. Apresentação de slides, http://www.ppt2txt.com/r/360ec66b/.

SJÖBERG, H.; YLINENPÄÄ, O. **Collision Detection for Haptic Rendering**. 2009. Dissertação (Mestrado em Ciência da Computação) — Umea University.

STANNEY, K. **Handbook of virtual environments**: design, implementation, and applications. [S.l.]: Lawrence Erlbaum Associates Publishers, 2002.

TRINDADE, M. R. M.; GRAZZIOTIN, R. U.; GRAZZIOTIN, R. U. **Eletrocirurgia**: sistemas mono e bipolar em cirurgia videolaparoscópica. ., http://www.scielo.br/scielo.php?pid=S0102-86501998000300010&script=sci_arttext.

ZILLES, C. B. **Haptic Rendering with the Toolhandle Haptic Interface**. 1995. Dissertação (Mestrado em Ciência da Computação) — Massachusetts Institute of Technology.