Mateus de Souza

Ferramenta para visualização de dados meteorológicos

Universidade Federal de Santa Catarina

Dezembro, 2011

Mateus de Souza

Ferramenta para visualização de dados meteorológicos

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Orientadores Prof. Dr. rer.nat. Aldo von Wangenheim

Co-orientador:
M.Sc. Tiago de Holanda Cunha Nobrega

Universidade Federal de Santa Catarina

Dezembro, 2011

Resumo

O presente trabalho apresenta o desenvolvimento de uma ferramenta para auxiliar a análise de eventors meteorológicos, através da visualizalção de campos 3D de dados obtidos a partir do modelo ETA gerado pelo INPE. A ferramenta servirá para estudo de casos por especialistas, bem como para a apresentação de dados meteorológicos para o público leigo. São utilizados métodos de visualização volumétrica de dados escalares e vetoriais, realizando a renderização através da placa aceleradora gráfica. É apresentado um algoritmo para geração de linhas de corrente para a visualização de dados vetoriais. Os dados também podem ser visualizados com a ajuda da realidade aumentada como meio de interação com o usuário. A interface criada possibilita a interação do usuário com uma grande quantidade de dados em tempo real, permitindo ações de rotação, translação e zoom da câmera, bem como outras funções para a inspeção dos dados.

Sumário

Lista de Figuras

1 Introdução

A meteorologia sempre se apoiou em instrumentos para auxiliar seu estudo. Desde os primórdios do estudo da atmosfera de nosso planeta, especialistas utilizaram instrumentos como barômetros, termômetros e pluviômetros para fazer medições das condições do tempo e com isso estudar o comportamento do clima em diferentes partes do mundo. Além de medir o estado atual, anotações eram feitas para gravar as mudanças climáticas e as variações com a passagem do tempo. Antes da invenção do computador, esses dados precisavam ser manipulados e arrumados manualmente pelos especialistas, para que esses pudessem tirar suas conclusões. Com esta invenção, tudo se tornou mais prático e usável. Agora é possível, além de manipular um número muito maior de dados, utilizar os computadores para fazer simulações fidedignas, utilizando modelos mais complexos.

Segundo Papathomas et al. [16], os tipos de dados meteorológicos disponíveis influenciam fortemente sua visualização. No início, as medições eram feitas apenas em 2D, adquirindo pouca resolução, de modo que os especialistas utilizavam-se de, por exemplo, linhas de contorno para mostrar campos bidimensionais. Com a captura de dados em 3D, os meteorologistas recebem agora a tarefa de perceber as estruturas volumétricas presentes na atmosfera. Para isso, alternavam entre várias imagens contendo as linhas de contorno de uma altura específica, para montar mentalmente uma imagem tridimensional dos dados. Finalmente, os dados passaram a chegar em uma taxa maior e os meteorologistas recebiam dados novos a cada hora, tornando possível a animação dos dados para que fosse possível um entendimento da evolução das condições do tempo.

Atualmente, os cenários analisados pelos especialistas podem ser bastante complexos. Por exemplo, em Novembro do ano de 2008 houve um período de muitas chuvas no estado de Santa Catarina. Foram 106 mortes confirmadas e 29 não confirmadas, mais de 1,5 milhões de pessoas afetadas, das quais mais de 5 mil ficaram desabrigadas [2]. As chuvas se estenderam por todo o mês de novembro, excedendo em muito o nível de precipitação esperado, e por uma área grande, atingindo a maior parte do estado. Por ser um evento de proporções tão grandes, torna-se de difícil estudo. É preciso analisar não só os dados meteorológicos do fim

das enchentes, mas acompanhar toda a evolução da catástrofe. E, com o aumento do número de dados relacionados, também aumenta a dificuldade em analisar o cenário como um todo. Esse é um cenário catastrófico muito estudado por meteorologistas , por ter causado as maiores enchentes na região nos últimos anos. Além disso, é de interesse geral da população obter informações sobre estes cenários catastróficos, para entender o que causou o desastre.

Torna-se necessária, então, a utilização de ferramentas computacionais para realizar a análise destes volumes de dados. Cenários como os descritos acima podem ser recorrentes em uma região do planeta, o que torna muito importante a sua análise, bem como da discussão dos cenários entre especialistas. Possuindo essas informações, o governo e a população local podem preparar-se para uma possível próxima ocorrência.

Para resolver este problema, é disposto no presente trabalho o desenvolvimento de uma ferramenta para visualização e análise de dados meteorológicos por especialistas, bem como para apresentação destes dados para um público leigo.

O resto desta monografia é dividido da seguinte maneira. Na próxima seção, a origem dos dados meteorológicos é explicada, bem como uma visão geral de como estes dados serão utilizados pelo usuário. Depois, são definidos as motivações, os objetivos e os cenários de uso, ainda neste capítulo. No Capítulo 2, é apresentada a revisão sistemática da literatura realizada para este trabalho. No Capítulo 3 são apresentados e explicados os métodos de visualização aplicados na ferramenta, bem como as decisões de projeto tomadas para o desenvolvimento dos algoritmos. Estes métodos então foram averiguados com a validação descrita no Capítulo 4. Finalmente, é apresentada a conclusão do trabalho, no Capítulo 5, e possíveis caminhos de pesquisa a serem seguidos, no Capítulo 6.

1.1 Variáveis Meteorológicas

Neste trabalho, como dito anteriormente, serão utilizados dados meteorológicos para realizar a análise dos cenários. Estes dados não são amostrais, mas provêm de modelos numéricos de previsão do tempo. Estes modelos conseguem simular o estado atual da atmosfera em determinada região do planeta. Além disso podem, como o nome sugere, prever a condição do tempo em um futuro próximo. Como saída destas simulações, temos um conjunto multidimensional de variáveis geofísicas. Para cada variável física, o modelo retorna um *grid* tridimensional contendo os valores desta variável na região sendo estudada.

Ao fim da previsão, o modelo provê variáveis *básicas* como temperatura da atmosfera, a umidade relativa do ar e a direção e velocidade do vento. A partir destas variáveis, é possível,

seguindo fórmulas matemáticas, calcular variáveis *derivadas* como gradientes, divergência, advecção e vorticiedade. Estas variáveis são agrupadas em um arquivo no formato GRIB (*Grid In Binary*). Estes dados são, então, extraídos do arquivo .GRIB e utilizados para popular um banco de dados. É a neste banco de dados que nossa aplicação vai buscar suas variáveis, assim como alguns metadados.

Além desta divisão entre variáveis *básicas* e *derivadas*, possuímos a distinção entre variáveis *escalares* e *vetoriais.* Escalares são as variáveis que possuem apenas um valor por célula do *grid* que representa o volume de dados proveniente do modelo numérico de previsão. Temperatura, umidade relativa do ar e magnitude do vento são exemplos de variáveis *escalares.* Vetoriais são as variáveis que possuem pelo menos dois valores por célula do *grid.* Estas variáveis representam direção, além da intensidade. Geralmente são formadas por três componentes: a intensidade nas direções X, Y e Z. Aqui cabem as variáveis que demonstram *movimento*, como o vento, gradientes de temperatura, etc.

1.2 Motivação e Justificativa

As ferramentas para visualização de dados meteorológicos, por sua origem científica e seu foco em precisão, podem ser de difícil manipulação para um usuário não-especializado. Além disso, com o aumento da complexidade (e quantidade) dos dados, é importante o desenvolvimento de uma ferramenta de fácil utilização, que torne intuitiva a visualização de dados tridimensionais. A interpretação dos dados ainda é difícil para leigos, mas um especialista poderia utilizar esta ferramenta em uma apresentação, por exemplo, onde ele explicaria os dados mostrados em tela para outros.

Em trabalhos de visualização, também, geralmente é aplicada uma técnica por vez. Isso motivou essa pesquisa na relação que diferentes modalidades de visualização formam quando integradas num mesmo ambiente.

Outra motivação é o aumento do número de aplicações que utilizam a realidade aumentada como auxílio na visualização. Esta técnica é muito utilizada em jogos, ou para ensino e treinamento, porém não encontra-se nenhum trabalho relacionado a dados meteorológicos.

1.3 Objetivos

Desenvolver uma ferramenta para facilitar a visualização de dados meteorológicos é o objetivo principal deste trabalho.

Para tanto, é necessário focar na *interatividade*. A interatividade aqui possui dois lados: a apresentação dos dados precisa ser interativa no sentido de que o usuário deve poder controlar os objetos na cena, bem como alguns parâmetros de visualização (posição da câmera, por exemplo), assim como deve ser interativa no sentido que o usuário possa obter resultados da aplicação em tempo real.

Também é importante, neste escopo, a *qualidade da visualização* gerada pelos algoritmos. A interatividade pode ter de ser sacrificada por uma visualização mais precisa e agradável. Esta é uma medida um tanto nebulosa de se quantificar, já que não há definição precisa do que é uma visualização “bela”.

A seguir são definidos alguns objetivos específicos que visam atender estas restrições, bem como a formalização do objetivo geral.

1.3.1 Objetivo Geral

Possibilitar a visualização interativa e tridimensional de dados meteorológicos escalares e vetoriais, criando um ambiente que auxilie a análise dos dados por meteorologistas. Integração de técnicas de visualização volumétricas com realidade aumentada, de modo a prender a atenção de usuários leigos e especialistas.

1.3.2 Objetivos Específicos

- Investigar diferentes métodos de visualização para dados volumétricos em um ambiente tridimensional, testando seus resultados e comparando-os.
- Desenvolver uma interface gráfica minimalista e funcional para a visualização dos objetos gráficos construídos com os métodos anteriores.
- Desenvolver e implementar um algoritmo para geração e visualização de linhas de correntes a partir de um campo vetorial.
- Realizar a integração de métodos de visualização diferentes para melhorar a informação que o usuário pode retirar de um volume de dados.
- Integrar os objetos criados pelos métodos anteriores com uma visualização auxiliada pela realidade aumentada, de modo a possibilitar uma interação natural com o usuário.

1.4 Cenários de Uso

Para ilustrar a utilização do programa desenvolvido, apresentamos alguns casos de uso.

Cenário de Uso 1

Um especialista deseja observar as condições meteorológicas de Santa Catarina durante as enchentes de 2008. Para tanto, ele abre a aplicação e seleciona um arquivo no banco de dados do dia específico que deseja estudar. Este arquivo contém todos os dados sobre o tempo daquele dia, como temperatura, umidade relativa do ar, velocidade e direção do vento. Após selecionar este arquivo, o especialista decide gerar uma visualização dos dados de temperatura, junto com a reconstrução de uma isosuperfície de umidade relativa.

O usuário interage com a visualização alterando o valor utilizado para a construção da isosuperfície. Também altera os valores utilizados pelas funções de transferência utilizadas na renderização dos dados. Finalmente, ele modifica a posição da câmera para encontrar um ângulo bom de visualização, de modo a salvar a imagem mostrada na tela para um arquivo.

Cenário de Uso 2

Um especialista deseja criar uma sessão para que possa, no futuro, rever as condições de tempo de um dia específico. Após fazer todas as manipulações necessárias e achar valores ótimos de reconstrução e visualização, o especialista salva a sessão de modo a carregá-la numa sessão futura. Primeiro, ele seleciona os filtros que deseja salvar (um filtro específico que diga as condições necessárias para a ocorrência de neve, por exemplo) e os salva na base de dados, sobre um nome específico. Depois, ele salva os objetos com seus valores de reconstrução, bem como a posição e configuração da câmera.

Depois de alguns dias, o especialista inicia o programa e carrega exatamente a mesma sessão que salvou dias atrás. Ele decide então comparar este cenário com um outro dia. Para tanto, ele abre o arquivo com os dados meteorológicos que descrevem o outro dia e aplica, sobre este novo conjunto de dados, os mesmos filtros que ele aplicou ao anterior, selecionando estes filtros da base de dados.

Cenário de Uso 3

De modo a verificar as condições do vento em uma região específica do estado de Santa Catarina, o usuário abre um arquivo de um dia escolhido por ele em uma base de dados. A partir

daí, ele pode fazer a reconstrução das linhas de correntes que descrevem o comportamento do vento naquela data. Ele move os planos de corte para achar a altura que deseja analisar.

O usuário ativa a realidade aumentada para fazer uma demonstração para leigos. Os dados são mostrados como se estivessem realmente na cena gravada pela câmera. O especialista explica então as condições demonstradas pelas linhas de correntes, utilizando a visualização como apoio à explicação.

Cenário de Uso 4

Um grupo de meteorologistas decidem analisar um cenário específico. Como estão separados geograficamente, eles criam uma sessão de teleconferência utilizando um programa externo que possua a habilidade de compartilhar a tela do computador do usuário (por exemplo, o programa Skype). Utilizando a ferramenta, eles abrem o modelo que descreve o cenário e utilizam as ferramentas para criar uma reconstrução significativa dos dados.

O especialista que está controlando a aplicação então ativa a realidade aumentada. Os outros especialistas agora, através da videoconferência, conseguem ver o primeiro meteorologista interagir com o volume de dados como se ele fosse um objeto real.

2 *Revisão Sistemática da Literatura*

Apresentamos neste capítulo a descrição de como foi realizada a revisão sistemática da literatura no contexto da visualização de dados meteorológicos. Seguimos os passos propostos por Kitchenham[12] para a produção de uma revisão imparcial e reproduzível: identificação da necessidade de uma revisão, desenvolvimento de um protocolo, identificação da pesquisa, seleção de estudos de interesse, validação da qualidade, extração de conhecimento, sintetização de dados e descrição da revisão.

2.1 Protocolo

No desenvolvimento desta revisão sistemática, criamos um protocolo seguindo o padrão definido por Kitchenham[12].

Para a realização desta revisão, formulamos as seguintes perguntas a serem respondidas:

- Como gerar uma boa visualização para linhas de corrente?
- Como se dá a interação entre a visualização simultânea de variáveis como temperatura, direção e velocidade do vento, umidade relativa do ar?
- A visualização simultânea é benéfica ao usuário? O aumento da informação disponível torna a análise dos dados difícil?
- Quais técnicas de visualização melhoram a compreensão dos dados meteorológicos?
- Como o usuário pode interagir com estes dados?
- O uso da realidade aumentada pode ajudar a visualização?

Com estas perguntas em mente, formulamos alguns conjuntos de palavras chave a serem utilizadas na busca por referências:

- (weather OU volumetric) data interactive visualization
- streamline (integration OU visualization OU seeding strategy)
- flow visualization
- augmented reality (visualization OU interaction)

Todas os conjuntos de palavras chave acima foram utilizados nos seguintes motores de busca:

- IEEExplore (http://ieeexplore.ieee.org/Xplore/guesthome.jsp)
- ACM Digital Library (http://portal.acm.org/)
- Google Scholar (http://scholar.google.com)
- Google (http://google.com)

2.1.1 Parâmetros de exclusão

Durante a realização da busca, é necessário aplicar alguns parâmetros para a exclusão de estudos primários desta revisão. As seguintes diretrizes foram as aplicadas na busca inicial, analisando o título dos artigos, seus resumos, bem como sua posição no resultado das buscas:

- Estudos que apresentassem um título não relacionado com o escopo definido pelas perguntas
- Estudos considerados não relevantes pelo motor de busca (somente os primeiros 50 primeiros resultados foram analisados)

Após o refinamento inicial dos resultados das buscas, eles foram analisados mais detalhadamente, levando em consideração a qualidade do trabalho e a sua relevância ao estudo.

Obras citadas frequentemente por outros estudos, mas que não apareceram na busca inicial, também foram incluídos na busca, pois geralmente formam uma base teórica para a área sendo analisada. Além destas obras, também foram incluídos estudos e ferramentas que possuíam credibilidade com os especialistas que auxiliaram na pesquisa. Estes estudos possuem especial relevância para nossa pesquisa, já que estas serão as ferramentas que os especialistas terão como referência para realizar uma validação de usabilidade.

Na próxima sessão serão mostrados os trabalhos submetidos a essa segunda fase da busca.

2.2 Resultados

2.2.1 Ferramentas para visualização de dados meteorológicos

Nesta seção são apresentadas algumas ferramentas de visualização específicas para dados meteorológicos. Apesar de o objetivo final destas aplicações ser bastante parecido com o da aplicação descrita nesta monografia, elas possuem características bastante distintas. Todas as ferramentas são desenvolvidas há bastante tempo, já sendo estabelecidas como boas ferramentas para visualização de dados meteorológicos.

McIDAS

Esta ferramenta vem sido desenvolvida desde o início da década de 70, sendo continuamente aprimorada até os dias de hoje. O McIDAS (Man Computer Interactive Data Access System) é desenvolvido em Java e permite a visualização, possivelmente animada, de dados meteorológicos adquiridos das mais diversas formas. Além da biblioteca para visualização, o McIDAS também possui procedimentos para análise, aquisição e gerenciamento dos dados.

A pesquisa começou na Universidade do Wisconsin-Madison, com o intuito de desenvolver uma ferramenta para previsão de tempo. Atualmente, o software já encontra em sua quinta geração, McIDAS-V, sucessor do McIDAS-X. É um *software* livre, *open source*, que possibilita a visualização de dados em 2D e 3D. Por ser baseado em Java, esta é uma solução multiplataforma.

Vis5D

Vis5D é um sistema computacional dedicado a visualizar dados na forma de um retângulo poli dimensional. Para ser mais exato, estamos lidando com 5 dimensões (daí o nome da ferramenta): as 3 dimensões de espaço, uma dimensão de tempo e uma dimensão para as diferentes variáveis meteorológicas. Esta foi a primeira ferramenta a produzir visualização interativa de dados 3D que variam com o tempo. Também foi o primeiro projeto *open source* de visualização tridimensional.

Este programa definiu um formato de arquivo próprio para a importar dados. O arquivo deve conter a descrição de todas as variáveis, em todas as dimensões de espaço e, possivelmente, uma de tempo. Além disso, é necessário descrever alguns metadados básicos, como suas dimensões, posição geográfica, orientação dos dados.

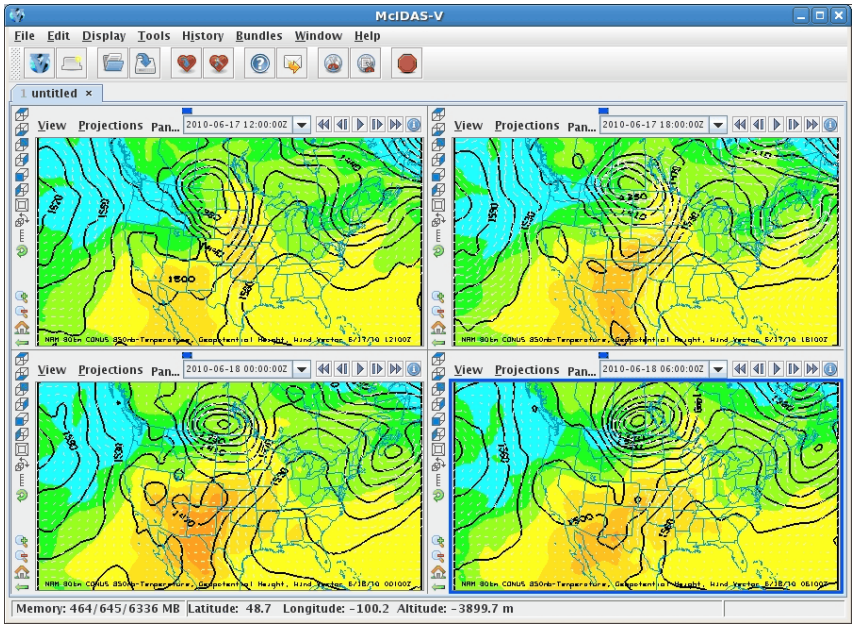


Figura 2.1: Uma *screenshot* do McIDAS-V visualizando dados de temperatura, altura geopotencial e vetores de vento

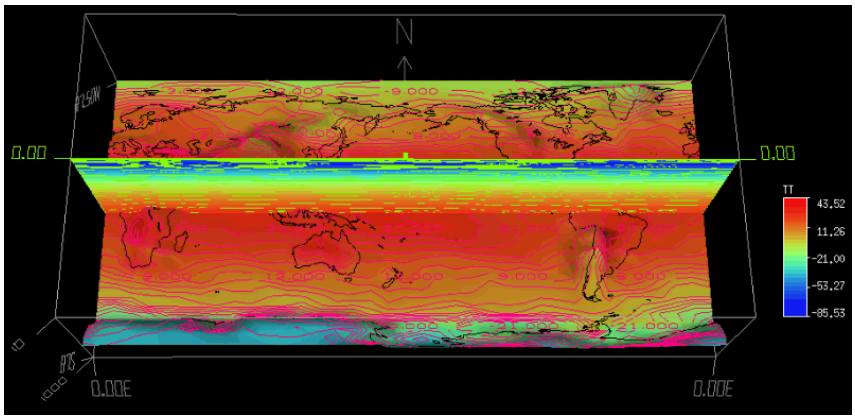


Figura 2.2: Vis5D visualizando dados de temperatura com planos de corte

Com o desenvolvimento da ferramenta, vários ramos foram criados. Para resgatar a unidade do projeto, foi criado o Vis5d+, que pretende unificar esses diferentes ramos dessincronizados.

GrADS

Diferentemente da ferramenta anterior, o GrADS realiza principalmente visualização em 2D. Porém, assim como o Vis5D, utiliza-se de um vetor de cinco dimensões para guardar os dados. Os *grids* de dados podem ser tanto regulares quanto irregulares.

O usuário controla a aplicação através de uma linha de comandos. Graças a isso, também é possível criar *scripts* para o GrADS, de modo a automatizar a criação de uma visualização específica. Além dos *scripts*, é possível a criação de rotinas externas às bibliotecas do aplicativo,

que podem ser escritas em diferentes linguagens de programação.

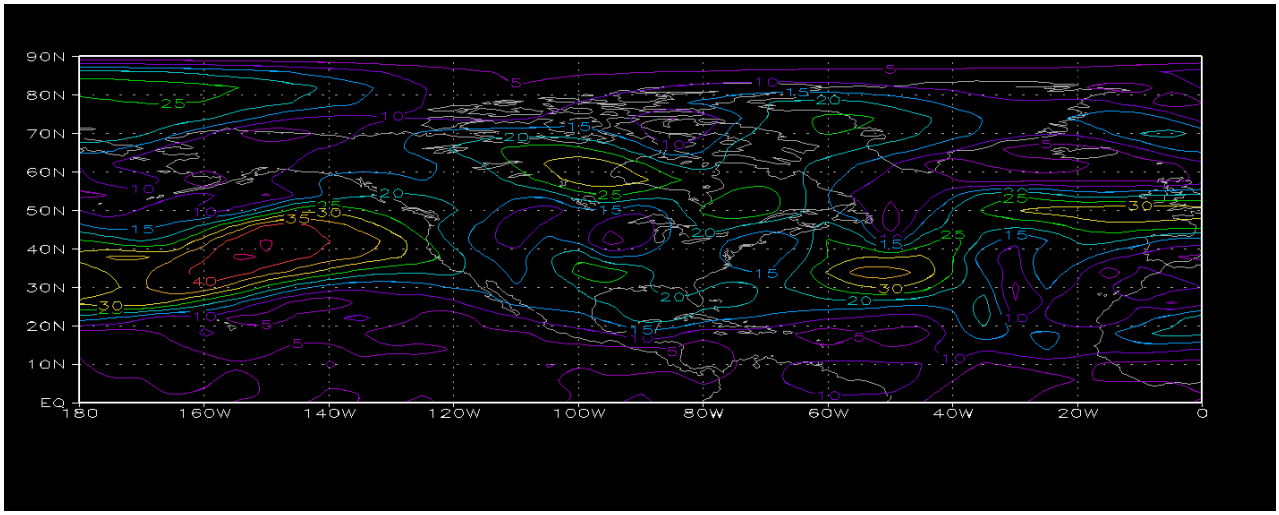


Figura 2.3: GrADS plotando linhas de contorno para a magnitude do vetor de vento

2.2.2 Visualização de dados escalares

Planos de Textura

Utilizar planos de textura é o tipo mais básico de visualização de volumes [8]. Nele, utilizamos planos de corte para visualizar uma fatia do volume por vez. Podemos manipular estes planos para explorar os dados.

A técnica é bastante simples, mas permite uma boa visualização em 2D de dados tridimensionais. O usuário pode se concentrar em uma parte de interesse do volume que deseja investigar. Um defeito desta técnica é que, por sua natureza bidimensional, ela não mostra claramente as estruturas dos dados. Ela também é focada em volumes uniformes.

Sendo muito utilizada áreas que remontam um volume a partir de várias imagens (visualização de imagens médicas, por exemplo), pode ser aplicado para exploração de volumes em geral.

Isosuperfícies

Esta modalidade de visualização visa construir uma superfície que passe por todos os pontos no volume que possuam um valor qualquer fornecido pelo usuário. Para isso precisamos de uma técnica que gere superfícies baseada nos valores dos voxels.

A solução encontrada foi o método conhecido como Marching Cubes [13]. Este algoritmo baseia-se em pequenos cubos que “caminham” pelo volume, construindo um mesh de triângulos

utilizando os valores amostrados nos 8 vértices do cubo.

Raycasting

Uma alternativa aos métodos de visualização citados acima é o chamado raycasting, um método para visualização direta do volume de dados[8].

Utiliza-se uma função de transferência para mapear valores escalares para valores de cor e opacidade. O método consiste, então, de traçar raios que partem da câmera na cena para o volume, amostrando os valores dos voxels encontrados e fazendo a conversão desses valores para uma cor.

Tenta-se simular o método utilizado pelos olhos humanos para formar imagens, mas no sentido inverso: ao invés de os raios de luz chegaram à câmera, eles são lançados a partir dela.

Realidade aumentada

Em seu artigo, Wang, Lee e Fang [18] apresentam uma ferramenta para visualização de imagens de tomografia computadorizada utilizando realidade aumentada.

Uma das motivações dessa visualização é que, com a realidade aumentada, é possível tornar mais natural a interação com os objetos 3D renderizados do que com uma visualização convencional. Por exemplo, um usuário pode rotacionar o volume com mais facilidade e, também, fazer o posicionamento da "câmera" como se estivesse olhando um objeto real.

A realidade aumentada é principalmente considerada para melhorar a experiência que usuários retiram de jogos. Alguns jogos comerciais já utilizam esta técnica, sendo o The Eye of Judgement [1] o mais conhecido. Outras aplicações foram desenvolvidas para estudar a viabilidade da técnica como meio de ensino [10], reabilitação motora [7] e até tratamento de fobias [11].

2.2.3 Visualização e integração de linhas de corrente

Aqui estão reunidos os trabalhos sobre integração, visualização e estratégias para geração de sementes para linhas de corrente (do inglês, *streamline*). É de decisão comum de todos os trabalhos que uma visualização boa para linhas de correntes gera linhas alongadas e igualmente espaçadas, que cubram todo o domínio. É importante, no entanto, não utilizar um número muito grande linhas de corrente para que a visualização não fique com muita informação desnecessária. Os algoritmos propostos a seguir tentam solucionar estes problemas.

Similarity-Guided Streamline Placement with Error Evaluation [6]

Este algoritmo define uma relação de semelhança entre as linhas de correntes para realizar a integração e, consequentemente, a eliminação de linhas de corrente desnecessárias.

A métrica de distância entre duas linhas é dividida em dois tipos:

- distância real: real distância entre duas *streamlines*. Quando mais longe os pontos de referência das duas linhas estiverem, maior é este parâmetro;
- forma e direção: leva em consideração a forma de duas *streamlines*, bem como sua direção.

Aqui é considerado que duas *streamlines* podem ter todos seus pontos coincidentes, mas possuírem direções diferentes, como é mostrado na Figura 2.4. Na imagem, (a) mostra duas *streamlines* com formas e direção muito parecidas, (b) com formas diferentes, mas mesma direção e (c) com forma e direção diferentes.

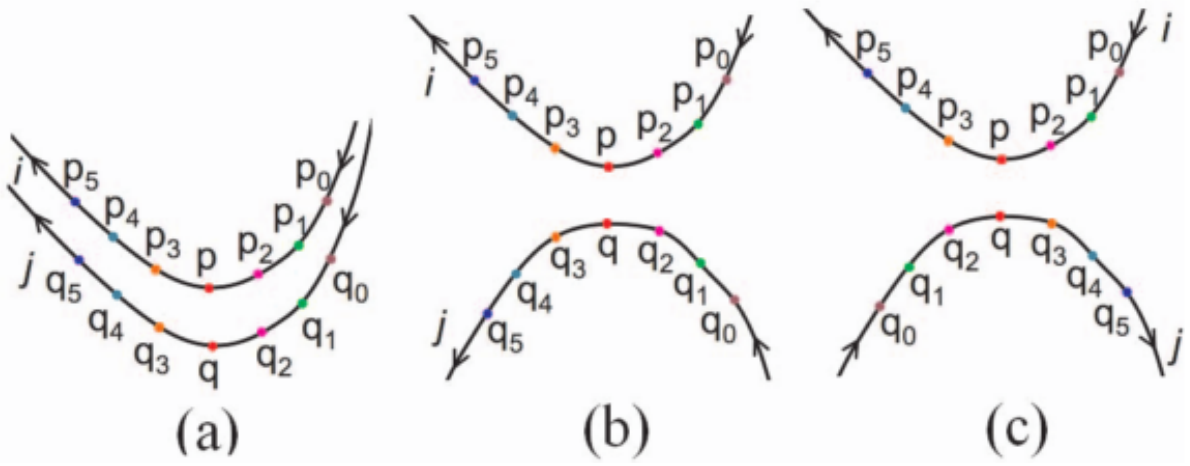


Figura 2.4: Semelhança entre linhas de corrente. Fonte: [6]

O importante desta técnica é elimiar linhas de corrente que não adicionem novas informação á visualização. Isto é feito eliminando linhas de corrente que sejam muito similares. Este limiar de similaridade pode ser ajustado para as necessidades da aplicação e não precisa ser contínuo no domínio. Ou seja, é possível ter, por exemplo, mais *streamlines* em lugares onde a magnitude dos vetores é maior.

O artigo não mostra nenhuma prova concreta do porquê, mas aparentemente esta técnica também realça as características topológicas do domínio. Perto de pontos críticos, as *streamlines* costumam ser muito distintas, enquanto em campos homogêneos, as *streamlines* se tornam

muito parecidas. Assim, lugares que precisam de mais *streamlines* para demonstrarem todas as suas características recebem mais *streamlines*.

Farthest Point Seeding for Efficient Placement of Streamlines [14]

É apresentado uma nova ideia para a geração de sementes: o ponto mais distante de todos os outros pontos já integrados é selecionado como candidato para ser a semente da próxima linha de corrente. Para fazer a aceleração do cálculo do ponto mais distante, uma triangulação de Delaunay é utilizada. Cada ponto novo integrado é inserido na triangulação. Para saber o ponto exato que será utilizado como próxima semente, pega-se o triangulo com maior área da triangulação e calcula-se o seu ponto médio geométrico.

A integração de uma linha de corrente acontece até o ponto em que ela colide com outra linha de corrente, consigo mesma ou com a borda do modelo sendo representado. Foi utilizado um integrador de Euler de primeira ordem, ou o de Runge-Kutta de segunda ordem.

Como o ponto mais distante é utilizado, é favorecida a criação de linhas de corrente alongadas. Não há garantia, no entanto, que as características do modelo sejam bem representadas.

A Streamline Placement Method Highlighting Flow Field Topology [19]

Neste método, após a criação de uma linha de corrente, o espaço é dividido em um ou mais sub-domínios. Em seguida, é achado um ponto para ser semente dentro de cada sub-domínio. A integração acaba quando não existe mais nenhuma ponto disponível para se tornar semente.

As sementes são selecionadas como o ponto médio do sub-domínio. É possível que este ponto esteja localizado fora do sub-domínio. Quando isto acontece, é escolhido um ponto dentro do sub-domínio utilizando retas que passam pelo ponto médio.

Para a integração de *streamlines*, é utilizado o integrador de Runge-Kutta de quarta ordem. As *streamlines* geradas não são igualmente espaçadas.

Pontos críticos não são diretamente calculados, mas são ressaltados naturalmente pelo algoritmo.

3 *Metodologia*

Neste capítulo serão detalhados os métodos escolhidos para a visualização dos dados meteorológicos.

Para a visualização de variáveis escalares, dois métodos foram escolhidos. O primeiro, a Visualização Multiplanar, é a mais simples das visualizações geradas pela ferramenta. Esta técnica é explicada na Seção 3.1. O segundo, a visualização por Isosuperfícies, gera uma superfície tridimensional que represente um valor específico da variável sendo estudada, e é detalhado na Seção 3.4. A visualização de variáveis vetoriais é feita através do método de linhas de corrente, descrito na Seção 3.3. A última seção deste capítulo define as técnicas utilizadas para a implementação da realidade aumentada na ferramenta.

Os dados foram mantidos em um banco de dados SQLite, de modo que o programa pudesse acessá-los facilmente. Todos os métodos descritos neste capítulo foram implementados utilizando a linguagem C++, além de utilizarem a biblioteca OpenGL para realizar a renderização. A interface gráfica, também implementada em C++, utilizou a biblioteca Qt.

3.1 Visualização Multiplanar

A primeira técnica de visualização a ser apresentada é também a mais simples: visualização multi planar. Os dados possuem uma natureza tridimensional, porém realizar visualização em três dimensões é consideravelmente mais complexo do que em um ambiente bidimensional (a Seção 3.4 mostrará um método de visualização tridimensional). Além disso, esta técnica serve apenas para dados escalares. Algumas propriedades dos dados vetoriais podem ser visualizadas (por exemplo, a velocidade do vento em determinado ponto), mas para visualizar dados vetoriais de maneira mais intuitiva utilizaremos um método apresentado na Seção 3.3.

Como o próprio nome indica, esta técnica utiliza vários planos navegando pelo volume para mostrar suas características. Estes planos geralmente são alinhados aos eixos do volume e possuem navegação limitada a um eixo. A posição do plano dentro do volume é utilizada para criar uma textura que representa aqueles pontos.

Este modo de visualização é muito utilizado, mas geralmente a textura 2D não é apresentada em um ambiente tridimensional. A natureza dos dados é pouco importante, desde que se possa montar um volume a partir deles. Aplicações de visualização de dados médicos, por exemplo, usam essa técnica para mostrar os dados de ressonâncias magnéticas e tomografias computadorizadas.

A navegação dos planos é contínua dentro do volume, apesar de os pontos de amostragem serem discretos. Ou seja, é possível colocar o plano *entre* diferentes níveis do volume. A visualização é gerada a partir de uma interpolação entre os valores dos *voxels* nos níveis adjacentes aos pontos do plano.

Com a visualização dos planos em 3D, o usuário tem uma noção direta de onde dentro do volume os planos se encontram. Apesar disso, estruturas tridimensionais do volume podem ficar obscurecidas pela visualização bidimensional.

Na Figura 3.1 podemos ver um exemplo de visualização multi planar. Os planos podem ser movimentar livremente por seu eixo. O volume representado na imagem possui os dados de temperatura na região de Santa Catarina em agosto de 2005. Como podemos averiguar na visualização, as cores quentes representam temperaturas elevadas (baixas altitudes) enquanto cores frias representam temperaturas baixas (altas altitudes). Com a visualização em três dimensões, o usuário já possui alguma noção da altura em que o plano horizontal está posicionado.

Nesta imagem também está mostrada a caixa que delimita o volume. Conhecida como *bounding box* (caixa delimitadora), esta caixa também restringe os movimentos dos planos,

mantendo-os sempre em seu interior.

Na imagem, também, é possível notar o exagero vertical aplicado. Dados de meteorologia possuem uma diferença grande de proporção entre a largura e comprimento do volume com a sua altura. Na imagem, o volume apresentado é de 95x81x19 voxeis. Ou seja, as dimensões de largura e comprimento são mais do que quatro vezes maiores que a dimensão de altura. Se a visualização fosse feita diretamente sobre este volume, seria difícil perceber as variações de temperatura ao longo da altura do mesmo. Por isso, aplicamos um exagero vertical neste volume. Esta técnica consiste em *esticar* o eixo vertical do volume, de modo a torná-lo mais regular. Neste caso específico, estamos utilizando um *voxel* de 1x1x2.

3.2 Função de Transferência

Os valores amostrados do volume são escalares. Ou seja, eles possuem apenas uma informação: o valor da variável sendo amostrada naquele ponto do volume. Precisamos, então, transformar estes valores em informação de *cor* para montar uma visualização a partir dos dados. Para tanto, fazemos uso de *funções de transferência.*

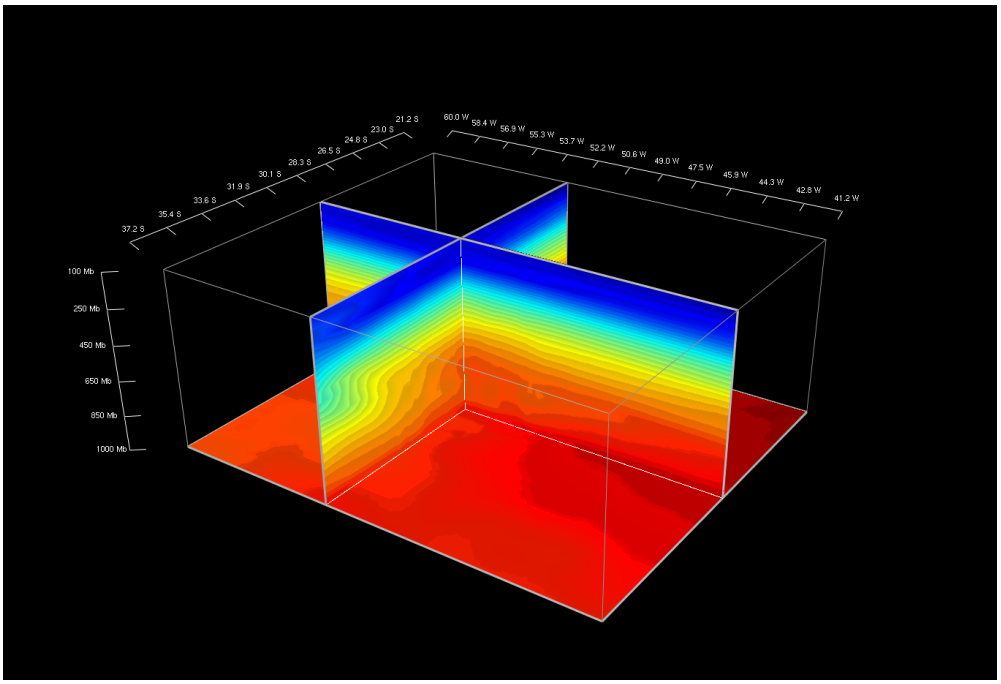
Estas funções associam um valor escalar com uma cor. A função mais simples, por exemplo, é a que mapeia todos os valores do domínio para uma única cor. A partir daí, podemos criar outras funções quaisquer, definindo sempre quais valores serão mapeados para cada cor. A interpolação de um escalar x é dada pela equação 3.1, onde c representa a cor associada ao escalar, a e b são os pontos que cercam x (sendo a o mais próximo dos dois) e a variável d representa o quão próximo de a x está.

$$c = color(a)(1-d) + color(b)d \tag{3.1}$$

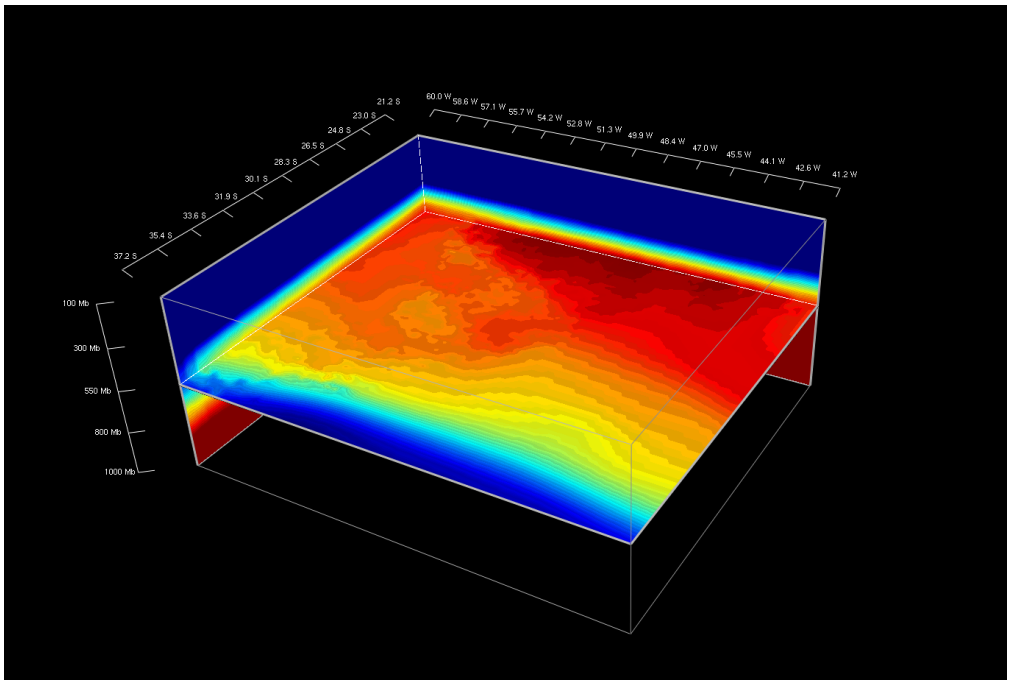
$$d = \frac{a-x}{a-b} \tag{3.2}$$

Em nossa aplicação, utilizamos apenas alguns pontos de controle para definir as funções. Pontos que estejam entre os pontos de controle recebem sua cor a partir de uma interpolação linear das cores nos pontos de controle. Assim, podemos definir funções de maneira rápida, sem precisar definir os valores de transferência para cada valor do domínio. Além disso, dada a natureza contínua de nossos dados, a definição de uma associação completa entre valores e cores seria impossível.

Apesar de estas funções serem bastante flexíveis, em nossa aplicação restringimos sua edi-



(a)



(b)

Figura 3.1: Visualização multi planar

ção, em tempo de execução, para a modificação dos valores máximos e mínimos do domínio. O mapeamento das cores é definido anteriormente em arquivos texto. Durante a visualização, apenas distribuímos esse intervalo no domínio da variável.

Na figura 3.1, diferentes funções de transferência são utilizadas para o mesmo volume de dados. A figura 3.1a mostra uma função de transferência com máximo em 25 °C e mínimo em -73 °C. Utilizando o mesmo arquivo, porém mudando o máximo para -3 °C e o mínimo para -27 °C, obtemos a figura 3.1b. Assim, o usuário pode alterar estes parâmetros para realçar uma faixa de valores específica com a qual ele queira trabalhar.

Alguns dados escalares possuem informação de *direção.* Tomemos, por exemplo, a variável que descreve a velocidade do vento zonal. Para esta variável, valores positivos denotam que o vento está indo para *leste*, enquanto valores negativos apontam para o *oeste.* Portanto, temos um ponto especial no valor *zero.* Para tornar a visualização mais intuitiva, desejamos que valores *acima* de zero sejam visualizados de maneira diferente que valores *abaixo* de zero. Para tanto, criamos uma função de transferência específica para estes casos. A ideia geral é que teremos *duas* funções de transferência para estas variáveis, uma para valores positivos e outra para valores negativos. Podemos então mudar o máximo e o mínio do domínio para cada uma das duas funções.

3.3 Linhas de Corrente

Nas seções seguintes é definido o algoritmo criado para a geração de linhas de correntes. Existem duas etapas principais deste algoritmo: integração de uma *streamline* e escolha de uma nova semente. Diferentemente do método descrito na Seção3.1, esta técnica é utilizada para gerar visualizações para campos vetoriais.

3.3.1 Algoritmo

O algoritmo desenvolvido utiliza-se de algumas ideias descritas nos trabalhos obtidos pela revisão sistemática apresentada no Capítulo 2. Para tornar a implementação mais fácil e incremental, algumas simplificações foram feitas até o estado atual. A escolha de sementes, por exemplo, tornou-se aleatória. Também por simplicidade, este algoritmo prevê apenas um campo vetorial bidimensional, apesar de possuirmos dados de campos vetoriais tridimensionais.

No Algoritmo 1 é possível ter uma visão geral de como uma *streamline* é gerada. Esta função recebe um ponto que será utilizado como semente, bem como um campo de colisão, onde testaremos se a *streamline* colide com alguma outra já integrada. A implementação deste campo de colisão será discutida na Subseção 3.3.3. A partir da semente, utilizamos o nosso integrador para gerar mais dois pontos, que passarão a fazer parte da *streamline.* A integração também será explicada mais a fundo na Subseção 3.3.5. É importante notar aqui, no entanto, que estamos fazendo integração nos dois sentidos da *streamline*, de modo a tentar torná-las mais alongadas. A primeira integração é feita utilizando o ponto semente. Após isso, utiliza-se as pontas da *streamline* para continuar o cálculo.

Algoritmo 1 Integração de uma *streamline*

```
Streamline generateStreamline( Point seed, CollisionField
    collisionField):
        Streamline streamline
        streamline.addFront( seed )
        while( ! collisionField.collides( streamline ) )
                Point forward = Integrator.forwardIntegration(
                    streamline.front() )
                Point backward = Integrator.backwardIntegration
                    ( streamline.back() )
                streamline.addFront( forward )
                streamline.addBack( backward )

        return steamline
```

A criação de um conjunto de *streamlines* a partir de um campo vetorial é descrita no Algoritmo 2. Este procedimento recebe um campo vetorial como parâmetro, assim como a distância de separação entre as linhas de corrente. Este segundo parâmetro é muito importante por definir de maneira indireta o número de *streamlines* que serão geradas. Quando o cálculo estiver terminado, retornamos uma lista contendo todas as linhas de corrente geradas. Primeiramente, criamos um campo de colisão para ser utilizado no método *generateStreamline*, descrito anteriormente. Além disso, o campo de colisão possui uma outra funcionalidade: indicar qual é o próximo ponto a ser usado como semente. Enquanto o campo de colisão possuir um ponto que pode ser utilizado como semente, pegamos este ponto e geramos uma nova linha de corrente a partir dele. Finalmente, atualizamos o campo de colisão para conter agora a *streamline* que acaba de ser gerada. Quando não houverem mais pontos que possam ser semente, todo o campo vetorial já foi cobrido por alguma *streamline*, portanto o processo pode acabar. Retornamos então a lista de todas as linhas de corrente geradas.

Algoritmo 2 Geração de um conjunto de *streamlines*

```
List<Streamline> streamlinesFromVectorField( VectorField
    vectorField, Double dsep ):
        List<Streamlines> streamlines
        CollisionField collisionField( vectorField.dimensions,
            dsep )
        while( collisionField.hasSeed() )
                Point seed = collisionField.nextSeed()
                Streamline newStreamline = generateStreamline(
                    seed, collisionField )
                streamlines.add( newStreamline )
                collisionField.update( newStreamline )

        return streamlines
```

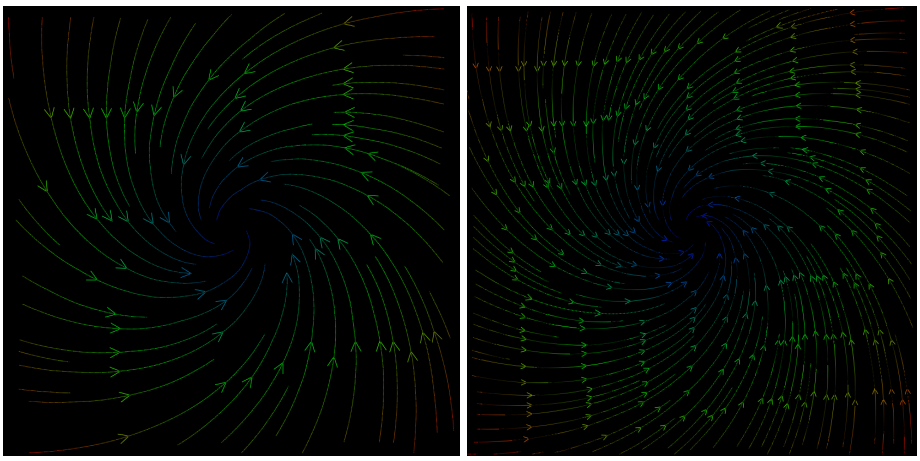
De modo a facilitar o entendimento do algoritmo geral, ele possui uma simplificação: a falta de auto-colisão. Em outras palavras, uma *streamline* não colide com ela mesma. Para resolver este problema, a atualização do campo de colisão foi movida para o procedimento descrito no Algoritmo 1. Cada ponto integrado que não gera uma colisão é adicionado ao campo. Assim, um novo ponto integrado para uma *streamline* vai colidir com os pontos que a formam.

Nas próximas seções são descritas com maior detalhe cada uma das partes deste algoritmo. Também são discutidos alguns parâmetros e sua influência na qualidade do resultado.

3.3.2 Distância de Separação

A distância de separação (algumas vezes abreviada para d_{sep}) é um parâmetro muito importante para este algoritmo. Ele dita o quão perto uma *streamline* pode chegar de outra. Assim, este parâmetro também define a quantidade de *streamlines* geradas. Por isso, é um dos principais parâmetros a ser controlando quando tentando gerar uma visualização que represente o campo vetorial sem deixá-la poluída.

Dois dos principais passos do algoritmo são afetados por este parâmetro diretamente: a integração das *streamlines* e a detecção de colisão. A integração, como veremos mais a frente, utiliza este parâmetro para gerar novos pontos para uma linha de corrente. Como o próprio nome diz, este parâmetro diz o quão longe uma das outras as *streamlines* devem estar. Assim,



(a) (b)

Figura 3.2: Exemplo de diferentes distâncias de separação

detectamos uma colisão quando duas *streamlines* estiverem mais próximas que este limiar.

A Figura 3.2 demonstra visualizações geradas com diferentes distâncias de separação para um mesmo campo vetorial. Na esquerda, podemos ver que um valor de separação alto gera menos *streamlines.* A imagem na direita mostra um valor de separação menor, mostrando mais detalhes do campo vetorial e tornando a visualização um pouco mais poluída. Para alguns campos vetoriais, pode ser necessário utilizar uma distância de separação menor para conseguir capturar todas suas características.

3.3.3 Campo de Colisão

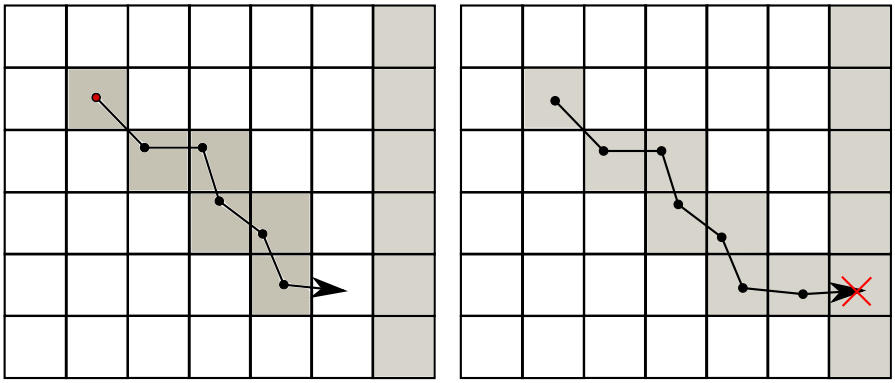
Uma das principais partes do algoritmo é a detecção de colisão. Aqui este termo se refere a detectar quando uma *streamline* colide com outra, de modo a parar sua integração. Como dito anteriormente, também estamos preocupados em detectar auto-colisões.

Para tanto, mantemos um *grid* que representará o campo vetorial. Cada célula deste grid possui largura determinada pela distância de separação, discutida anteriormente. A detecção de colisão é feita verificando o *status* de ocupação da célula que contém um novo ponto. Se já existir um ponto dentro da célula, há uma colisão. Nosso método de integração garante que um novo ponto integrado está a uma distância constante do último ponto, que é igual ao parâmetro de distância de separação das linhas de corrente. Assim, garantimos que não haverá casos em que um ponto integrado atravesse uma célula ocupada.

Esta estrutura foi criada para prover agilidade ao algoritmo. A detecção de colisão é uma das operações mais realizadas, juntamente com a integração. Graças ao tempo de execução

constante deste método, esta etapa não causa um detrimento no desempenho do algoritmo. No entanto, o gasto de memória pode se tornar um fator de risco quando a distância de separação tornar-se muito pequena em conjunto com campos vetoriais grandes.

A Figura 3.3 demonstra o *grid.* A integração de uma *streamline* começa de sua semente (o ponto vermelho na figura), seguido de vários pontos integrados. A cada ponto válido novo, o status da célula que o contém é atualizado para ocupado (células escuras na figura). Na Figura 3.3a, temos uma *streamline* sendo integrada. Podemos ver que ela se aproxima de outra linha de corrente. No próximo passo da integração (Figura 3.3b), o ponto integrado está dentro de uma célula já ocupada. Com a colisão detectada, o ponto é excluído da *streamline* atual e a integração é concluída.



(a) (b)

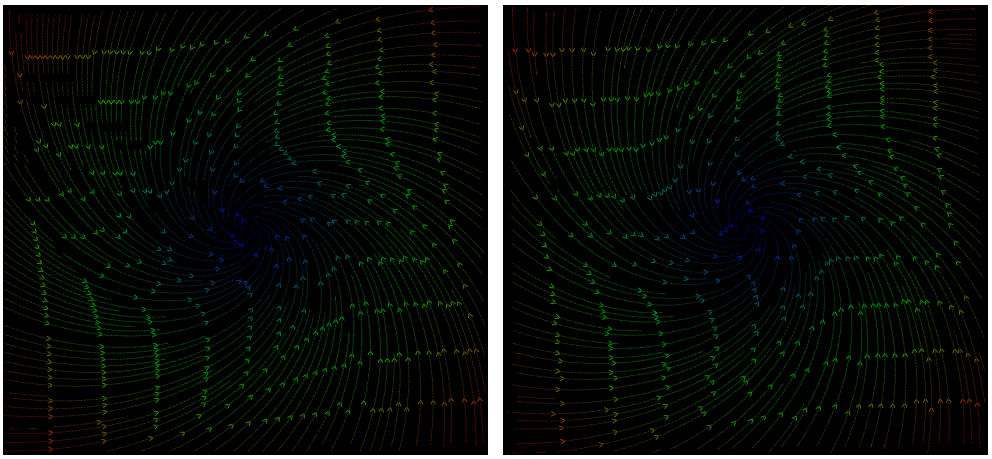
Figura 3.3: Demonstração do campo de colisão. O ponto em vermelho indica a semente da *streamline*, seguida dos próximos pontos integrados.

3.3.4 Escolha de sementes

Por onde começar a integrar uma *streamline* é uma escolha fundamental ao algoritmo. Pontos específicos podem melhorar a visualização e diminuir o número de artefatos. Por simplicidade, nosso método escolhe aleatoriamente um ponto dentre os disponíveis. Apesar de não garantir uma visualização que represente todos os aspectos do campo vetorial de maneira satisfatória, essa abordagem diminui o número de artefatos na visualização.

No começo do desenvolvimento, as sementes não eram escolhidas aleatoriamente. Isso fazia com que a geração de linhas de corrente fosse determinística, sempre gerando a mesma visualização, mas também criava uma visualização com *streamlines* curtas. A Figura 3.4 demonstra os dois tipos de escolha de semente. Na Figura 3.4a, podemos ver as *streamlines* geradas por um algoritmo determinístico. A ordenação das células para escolha como semente

faz com que as primeiras escolhidas sejam as localizadas no canto inferior direito. Geralmente as linhas que são integradas ao fim do processo são de menor tamanho, pois que o espaço já foi preenchido por *streamlines* anteriores, causando mais colisões. A forma determinística de escolha de sementes agrupa essas *streamlines* finais no canto superior esquerdo da tela, o que gera uma visualização pobre. A Figura 3.4b mostra o resultado utilizando a forma atual de escolha de sementes. Podemos perceber que as linhas de corrente são mais uniformes, tanto em comprimento quanto em cobertura do campo vetorial. A deficiência deste método é que o resultado da geração de *streamlines* pode não ser o mesmo para um campo vetorial em diferentes execuções.



(a) (b)

Figura 3.4: Diferentes métodos de escolha de sementes

O campo de colisão agrega a funcionalidade de escolher a próxima semente a ser usada. O motivo desta escolha é que o campo de colisão já possui a informação de quais células já estão ocupadas. Para tomar uma nova semente, escolhemos aleatoriamente uma célula não ocupada e utilizamos seu centro.

3.3.5 Integração

Para a geração das linhas de correntes, é necessário fazer a integração de novos pontos a partir da semente. Integração, no nosso contexto, pode ser entendida como a simulação de como uma partícula sem massa se moveria pelo campo vetorial, se este se mantiver constante. Tomamos um ponto semente como o início deste movimento e simulamos o movimento desta partícula utilizando os vetores do campo vetorial como forças que movem esta partícula. Estamos tentando aproximar, com isso, a trajetória desta partícula pelo campo vetorial, de modo a extrair as características que o definem.

O método mais conhecido para integração é o método de Euler, que é o atualmente utilizado como nosso integrador. Este método utiliza o Teorema de Taylor para a aproximação dos valores de uma função. Segundo Taylor, o valor de uma função pode ser aproximado pela série:

$$f(x+\Delta x) = f(x) + \triangle x f'(x) + \frac{\triangle x^2 f''(x)}{2!} + \frac{\triangle x^3 f'''(x)}{3!} + ... \tag{3.3}$$

O método de Euler consiste então em utilizar somente os primeiros dois termos desta fórmula para fazer a aproximação. Assim, simplificando a equação 3.3, obtemos a seguinte equação:

$$f(x+\triangle x) = f(x) + \triangle x f'(x) \tag{3.4}$$

Por desconsiderar os outros termos da série de Taylor, existe um erro que depende do tamanho de $\triangle x$ (chamado de passo de integração). Este valor encontrado para aproximar a função em um ponto $x+\triangle x$, é utilizado então para encontrar os próximos valores. Ocorre, portanto, uma acumulação do erro, ou seja, quanto mais longo for o processo de integração, mais provável é que o valor obtido seja diferente do valor real.

Tomando-se então um valor inicial para x e um passo de integração pequeno o suficiente, podemos aproximar uma função utilizando apenas sua derivada primeira. Para a integração de linhas de corrente, utilizamos o método acima para descobrir os valores da função de posição de uma partícula no campo vetorial, dado um ponto inicial qualquer. Os dados de direção e magnitude presentes no campo vetorial servem como derivada primeira da posição da partícula numa posição qualquer do campo.

Novamente, o parâmetro de distância de separação das *streamlines* toma um papel importante no algoritmo. Aqui, ele define o passo de simulação. Deste modo, a *magnitude* do vetor é ignorada em nossa integração, tomamos apenas a sua direção. A informação da *magnitude* não será perdida, sendo guardada para ser utilizada apenas na etapa de visualização. Para valores pequenos de d_{sep}, temos um aumento no tempo necessário para uma simulação. Como o passo de simulação é menor, demoramos mais tempo para atingir uma colisão. No entanto, o resultado da simulação tende a ser mais preciso, já que utilizamos mais pontos de amostra do campo vetorial para corrigir a trajetória da partícula na simulação, se ela estiver com erro.

Em nossa aplicação, geralmente é escolhido um valor mediano. Estamos tão preocupados com a interatividade da aplicação quanto com a qualidade da visualização. Valores muito baixos

de d_{sep} podem deixar o algoritmo lento, por aumentar o tempo de cálculo de cada *streamline.* Além disso, muitas vezes os usuários não desejam um nível de detalhe muito grande, estando mais preocupados com o comportamento geral do campo.

Nosso algoritmo prevê duas direções de integração para favorecer *streamlines* alongadas. Como em campos vetoriais retirados de dados reais (que são o objetivo de visualização de nosso método) os vetores mudam de valores de maneira contínua, podemos realizar um passo chamado de *backward integration* (integração para trás). Basicamente, estamos realizando a integração no sentido contrário do normal, utilizando um vetor oposto ao que amostramos.

Quando tomamos uma semente e realizamos a clássica *forward integration* (integração para a frente), seria como simular o movimento da partícula no sentido natural do tempo. Estamos avançando o tempo e utilizando a integração para descobrir em qual ponto esta partícula estará, dado o vetor de movimento do campo vetorial. Na *backward integration*, estamos indo no sentido contrário. Ou seja, em vez de integrarmos para descobrir para onde a partícula *vai*, estamos integrando para descobrir de onde ela *veio.* Podemos entender isso como uma integração no *outro* sentido do tempo.

Como dito anteriormente, esta integração é possível graças à continuidade de nossos campos vetoriais. Tomando valores de d_{sep}pequenos o suficiente, o erro de utilizarmos o vetor da posição atual para achar o ponto de onde a partícula *veio* é pequeno.

Em nosso algoritmo, a integração é interrompida somente quando a partícula simulada colide com uma *streamline* ou colide com as bordas do campo vetorial.

3.3.6 Visualização

Finalmente, após gerarmos todas as linhas de corrente de modo a retratar fielmente o campo vetorial, precisamos fazer com que sua visualização seja agradável, limpa e de fácil entendimento.

Nas imagens mostradas anteriormente (Figuras 3.2 e 3.4), é possível notar que as *streamlines* estão coloridas. Esta coloração mostra a magnitude do vetor que deu origem ao ponto em questão. Assim, quando visualizando um conjunto de linhas de corrente descrevendo as correntes de vento em uma região, podemos visualizar a *velocidade* do vento em cada ponto, além de sua direção e sentido, através de uma tabela de cores. Nas imagens mostradas anteriormente, cores quentes demonstram os vetores de grande magnitude, cores frias, os de baixa magnitude.

Além de mostrar a magnitude de nossas linhas, era necessário mostrar seu sentido. O re-

sultado de nosso algoritmo é uma fila de pontos, que representam para onde a linha de corrente está indo, a partir de sua origem. Mas se apenas desenharmos este pontos na tela, esta informação será perdida. Portanto, foram adicionadas setas para indicar o sentido das *streamlines*. Originalmente, utilizávamos triângulos no lugar de setas. Esta abordagem se tornava pouco clara quando o valor de separação era muito pequeno, por isso decidimos trocar o desenho das linhas de corrente para o estado atual, que pode ser visto nas imagens deste capítulo.

De modo a tornar a visualização mais agradável aos usuários, os pontos integrados para as *streamlines* não são visualizados diretamente. Criamos uma *spline* utilizando os pontos da linha de corrente como pontos de controle. Quando a distância de separação era muito grande, o número de pontos integrados por *streamlines* diminuía, o que gerava uma visualização descontínua. Utilizando esta curva, fazemos com que a linha de corrente seja contínua, independente do número de pontos integrados.

3.4 Isosuperfícies

Até o momento, todas as visualizações apresentadas neste trabalho utilizam apenas informações em duas dimensões. As informações de direção e velocidade do vento podem, na maior parte do tempo, utilizar apenas estas técnicas bidimensionais. No entanto, algumas estruturas presentes no volume podem passar despercebidas a menos que sejam destacadas com uma visualização tridimensional.

Para tanto, apresentamos uma técnica de visualização conhecida como *Marching Cubes* (cubos marchantes). Este método consiste em extrair uma isosuperfície do volume de dados, utilizando um cubo que percorre este volume.

Uma isosuperfície é definida como uma superfície dentro do volume de dados que possui, em todos os seus pontos, o mesmo valor escalar amostral. Este método, então, tenta construir uma malha de triângulos que descreva esta superfície de maneira precisa para um valor escalar qualquer. É importante notar que uma isosuperfície não é necessariamente contínua no volume. Ou seja, ela pode ser constituída de várias partes totalmente separadas no espaço.

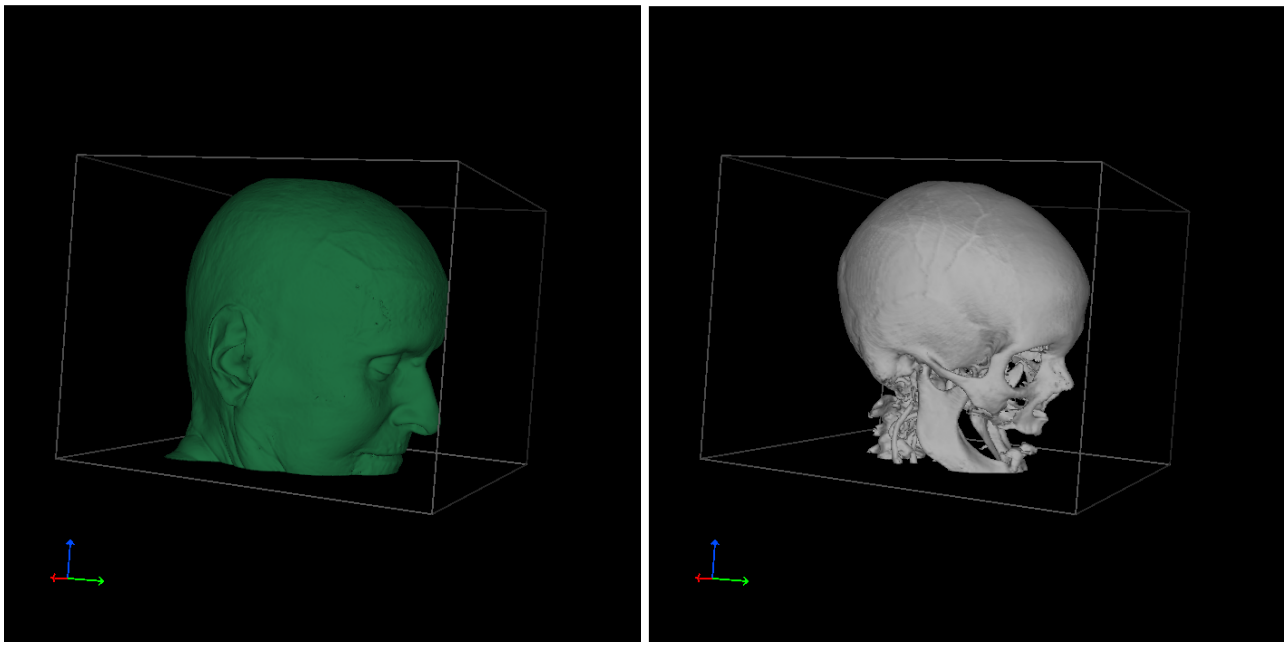
Apresentamos nas próximas seções uma visão geral da técnica, assim como uma descrição mais profunda de seu algoritmo. Ao fim deste capítulo, apresentamos algumas funcionalidades de visualização de isosuperfícies implementadas em nossa ferramenta.

3.4.1 Marching Cubes

A ideia de utilizar um cubo para percorrer um volume e amostrar valores de modo a construir uma malha de triângulos não é nova. Lorensen e Cline[13], publicaram o primeiro artigo sobre esta técnica em 1987.

O objetivo inicial desta técnica era criar uma representação tridimensional de estruturas do corpo humano a partir de imagens médicas, como podemos ver na Figura 3.5. A Figura 3.5a mostra uma reconstrução utilizando a densidade radiológica da pele como parâmetro, enquanto a Figura 3.5b toma a densidade dos ossos como parâmetro. O volume de dados é feito através do empilhamento de imagens médicas provenientes de exames como uma ressonância magnética. A partir deste volume, a técnica propunha reconstruir partes do corpo humano utilizando a densidade radiológica como parâmetro. Ao fim do processo, isosuperfícies de densidade radiológica eram construídas. A estrutura a ser reconstruída pode ser selecionada através da densidade radiológica, já que diferentes tecidos possuem densidades distintas.

A saída do algoritmo é um conjunto de triângulos que formam a superfície tridimensional,



(a) (b)

Figura 3.5: *Marching Cubes* utilizado em imagens médicas

que podem ser então visualizados pelo *pipeline* gráfico clássico. Um ponto importante desta técnica é que ela gera uma reconstrução 3D, que é independente da posição da câmera ou se configurações de visualização. Isto faz com que este método tenha certa vantagem sobre outras técnicas de visualização que geram apenas uma visualização (como *Raycasting*, por exemplo). Apesar de ser um método mais custoso, a saída do algoritmo pode ser manipulada e visualizada de diferentes maneiras, sem a necessidade de recalcular a isosuperfície.

Outro ponto deste algoritmo é que ele é altamente paralelizável. Este é um fator muito importante para aplicações gráficas, já que podemos utilizar o alto desempenho computacional das *Graphics Processing Units* (unidades de processamento gráfico) para obter um aumento do desempenho nos cálculos.

3.4.2 Algoritmo

O elemento básico (e o que dá o nome à técnica) é o cubo que percorrerá o volume. Os vértices do cubo são os pontos de amostragem do volume, ou seja, o cubo está *entre* duas fatias do volume. Percorremos o volume de maneira ordenada, duas fatias por vez, analisando os valores em cada vértice do cubo e construindo um conjunto triângulos baseado nestas amostras.

Em cada cubo, marcamos todos os vértices que possuam valor *maior ou igual* ao desejado para a construção. A isosuperfície, então, intersecta cada aresta do cubo que possua um ponto

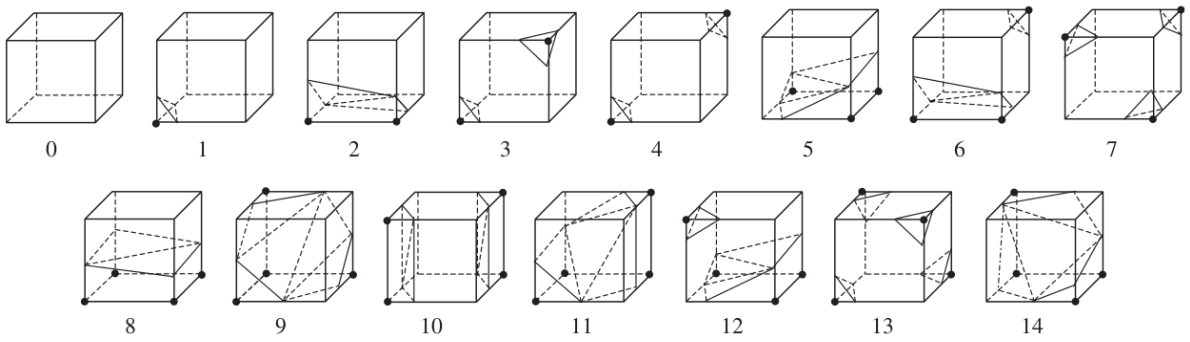


Figura 3.7: As 15 possíveis topologias de um cubo para o algoritmo. Fonte: [15]

marcado e outro desmarcado em suas extremidades. A estratégia geral é, dada a classificação de todos os vértices do cubo, descobrir qual conjunto de triângulos melhor representa a intersecção deste cubo com a isosuperfície.

O caso mais simples é quando todos os vértices do cubo possuem a mesma classificação. Deste modo, a isosuperfície não possui nenhuma intersecção com o cubo em questão, de modo que nenhum triângulo precisa ser construído.

Como possuímos 8 vértices por cubo, existem 256 combinações possíveis para classificarmos um cubo. Porém, algumas classificações são equivalentes. Duas classificações são ditas equivalentes se é possível, a partir de uma, alcançar a outra por meio de *rotações* e *reflexões* (Figura *3.6*). Rotação aqui se refere ao ato de rotacionar o cubo no espaço, enquanto reflexão denomina o ato de inverter a classificação de todos os vértices de um cubo. Isso reduz o número de classificações possíveis para 15 (Figura 3.7).

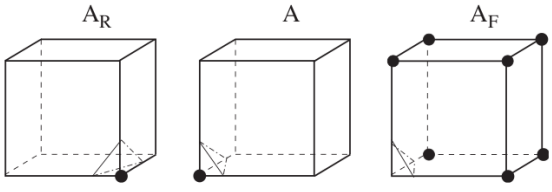


Figura 3.6: Imagem retirada de [15] demonstrando as simetrias reflexiva (A com A_F) e rotacional (A com A_R)

Após descobrirmos a classificação correta de um cubo, podemos realizar a construção dos triângulos (se necessário). Cada classificação já possui um *template* de intersecções a ser seguido, precisamos calcular o ponto correto onde a intersecção ocorre. Isto é feito com uma interpolação simples dos valores nas extremidades da aresta, de modo a descobrir o ponto que possui o valor da reconstrução. Finalmente, possuindo os pontos de intersecção, os triângulos

podem ser criados.

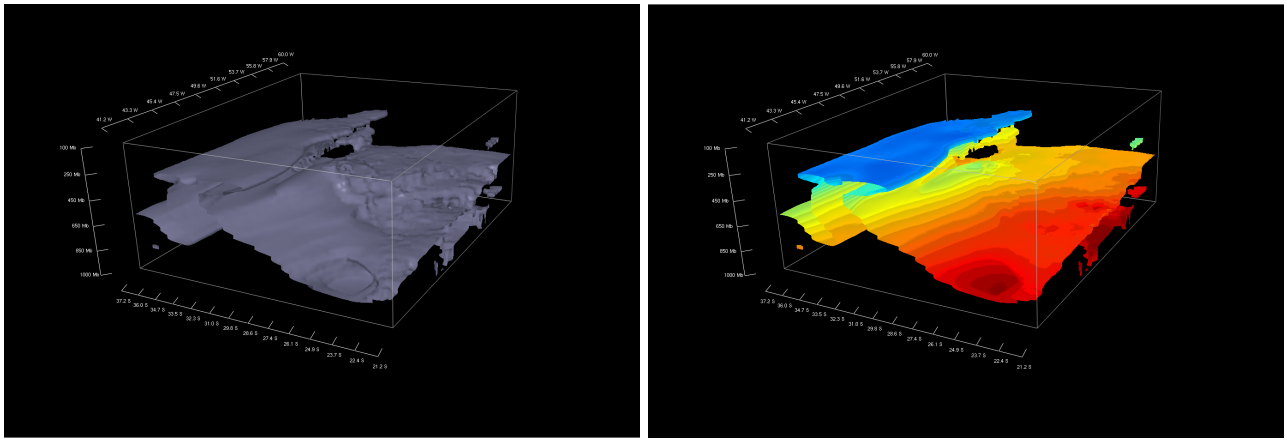
Um ponto importante de se notar nesta abordagem é que cada aresta só precisa ser calculada uma vez. Como a maioria das arestas pertence a mais de um cubo, isso pode diminuir drasticamente a quantidade de cálculos necessária. Para um cubo ao meio do volume (que não esteja em suas bordas), precisamos calcular apenas 3 novas arestas. Todas as outras arestas deste cubo em cubos anteriores. Esta otimização só é possível graças ao percorrimento em ordem realizado neste algoritmo.

O algoritmo descrito aqui é o *marching cubes* clássico, sem nenhuma extensão. Como esta é uma técnica relativamente antiga e bem estabelecida, várias extensões foram propostas para melhor ou o desempenho ou a abrangência desta técnica. Uma expansão bastante conhecida é o *Marching Tetrahedra [17]* que utiliza tetraedros no lugar de cubos para realizar a construção da malha de triângulos. Além disso, após a publicação deste algoritmo, foi verificada a existência de ambiguidades em algumas configurações que resultavam em falhas nas triangulações resultantes. As técnicas para eliminação destas ambiguidades estão resumidas em [15].

3.4.3 Visualização

Após a extração da isosuperfície do volume, precisamos visualizá-la. Como dito anteriormente, a isosuperfície resultante é composta por triângulos que podem ser visualizados pelo *pipeline* gráfico padrão, sem nenhuma alteração. Portanto, de modo a criar uma visualização simples, atribuímos apenas uma cor fixa para todos os triângulos e visualizamos a isosuperfície como uma malha preenchida.

Esta visualização, no entanto, não possui nenhum dado além dos pontos da isosuperfície em si. Podemos inserir mais informações nesta malha, de modo a tornar a visualização mais rica. Na aplicação, utilizamos uma variável meteorológica para construir a isosuperfície. Então, utilizando os pontos da superfície, podemos mostrar dados de outra variável, através da coloração dos triângulos. Para tanto, utilizamos uma função de transferência para mapear os valores da segunda variável para cores. Amostramos o volume de dados da segunda variável utilizando os pontos construídos a partir da primeira para achar os valores a serem passados para as funções de transferência.



(a) (b)

Figura 3.8: Resultado do algoritmo aplicado à dados de umidade relativa

3.5 Realidade Aumentada

Quando estamos visualizando dados tridimensionais, é preciso tornar fácil para o usuário manipular estes objetos. Neste trabalho, utilizamos a técnica de realidade aumentada para interação com os volumes, bem como para tornar a visualização mais interessante.

A realidade aumentada (do inglês *augmented reality)* é sempre comparada com sua irmã mais conhecida, a realidade virtual. Na realidade virtual, possuímos um ambiente totalmente sintético onde o usuário é imerso. Esta técnica é bastante popular, possuindo várias referências na cultura geral, desde filmes como *The Matrix* e *Tron* até livros como *Neuromancer.* Existe, no entanto, uma diferença marcante entre realidade *virtual* e *aumentada.* Em uma realidade *aumentada*, o usuário ainda possui contato com o mundo real. Ou seja, informações são *adicionadas* à realidade já apresentada ao usuário, de modo a tornar o mundo mais informativo ou interessante. A abordagem da realidade virtual seria *construir* um mundo sintético e inserir o usuário neste mundo.

Azuma [4] lista três componentes primordiais para realidade aumentada:

1. Combinação do mundo real com um virtual
2. Interação em tempo real
3. Três dimensões

Esta definição retira da categoria de realidade aumentada filmes como *Jurassic Park*, já que, apesar de misturar cenas reais com objetos virtuais feitos por computação gráfica, o filme não é uma mídia interativa. Geralmente, são utilizados *displays* específicos para as aplicações. Óculos especiais fazem com que o usuário veja apenas a cena com as informações já inseridas, tornando a interação mais intuitiva.

Esta técnica vem sendo utilizada cada vez mais atualmente, principalmente devido à evolução dos dispositivos móveis, que possibilita a aplicação da realidade aumentada em celulares e *tablets*. Jogos, principalmente, chamam a atenção dos usuários por mostrarem uma interação dos personagens virtuais com o mundo real. Exatamente por essa capacidade de prender a atenção do usuário durante a execução, a realidade aumentada é utilizada também no ensino através de jogos (o chamado *edutainment*).

Uma das fases mais importantes é a aquisição de informações sobre o mundo real, de modo que as objetos possam ser adicionados. Informações espaciais são necessárias para que os objetos virtuais pareçam realmente interagir com o mundo real. Várias técnicas são utilizadas

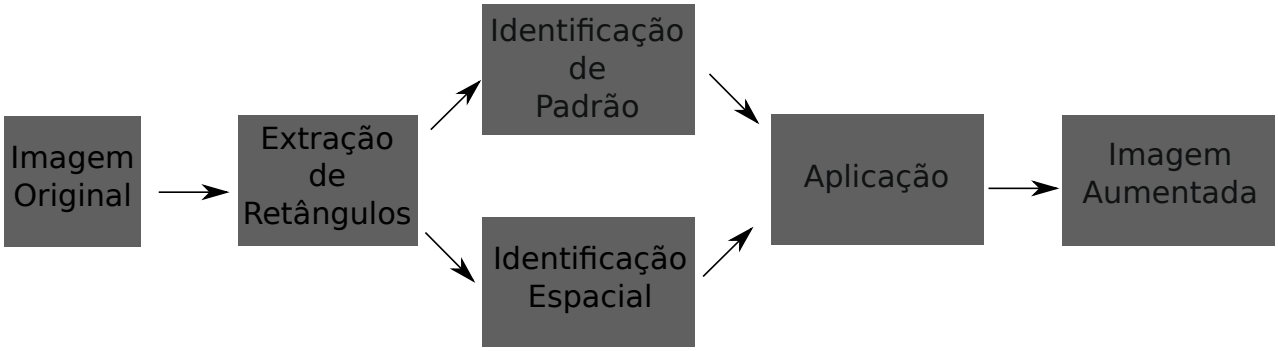
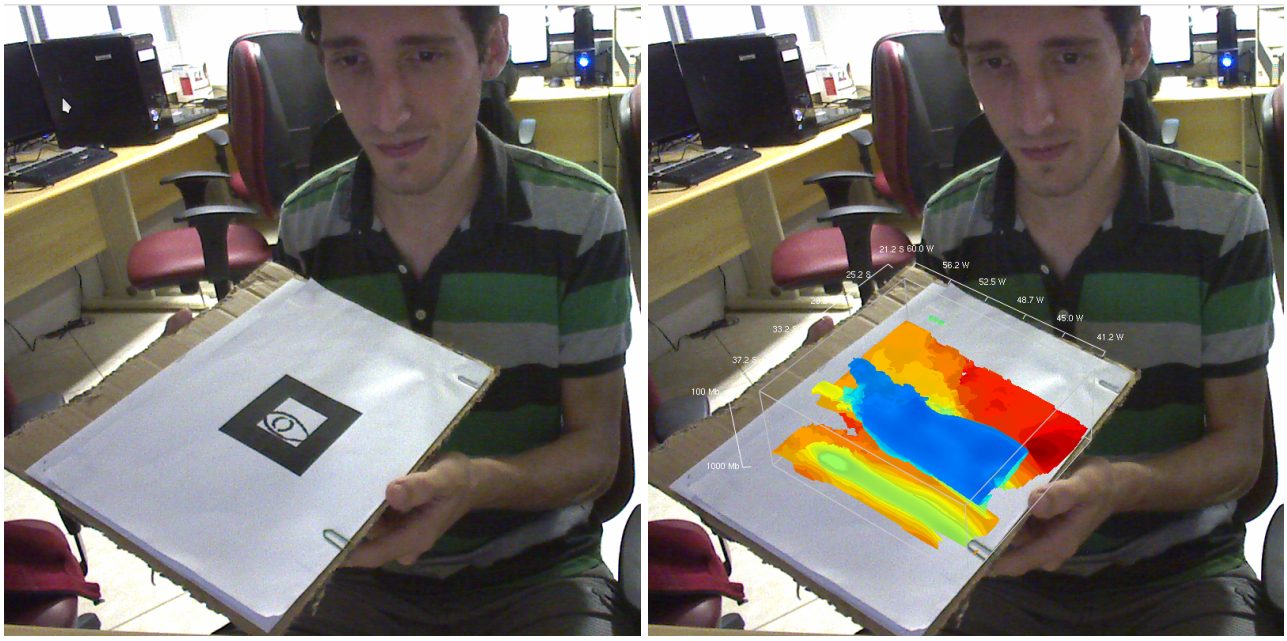


Figura 3.9: Passos de cálculo para a realidade aumentada

para obter os dados de onde posicionar os objetos virtuais, mas aqui focaremos na técnica que utiliza *markers*. Chamamos de *marker* um símbolo que é inserido no mundo real de modo a informar a aplicação da posição desejada de um objeto virtual.

Nesta aplicação, utilizamos o *framework* ARToolKit [3]. De maneira simples, o ARToolKit realiza todas as tarefas relacionadas a visão computacional e nos retorna uma matriz de transformação para a câmera da cena. Precisamos, então, realizar o desenho da nossa cena utilizando os parâmetros para câmera dados, para posicionar os objetos em seus *markers*.

A Imagem 3.9 mostra os principais passos da parte de realidade de nossa aplicação. O *framework* ARToolKit realiza todos os passos antes da Aplicação. De entrada, possuímos uma imagem capturada por uma câmera. A partir desta imagem, são descobertos retângulos que podem representar *markers*. Estes retângulos são extraídos da imagem e passados para as próximas etapas. Na *Identificação de Padrão,* é feito o reconhecimento do padrão interno ao *marker*. Com o ARToolKit, é possível utilizar mais de um *marker* numa mesma cena. A informação espacial dos retângulos (sua posição e orientação no espaço) são extraídas na *Identificação Espacial.* A informação de qual padrão está sendo visualizado e a orientação dos retângulos é então passada para a aplicação. Com isso, a aplicação realiza o desenho dos objetos no lugar certo do espaço e mostra ao usuário uma *Imagem Aumentada*, já com as informações adicionadas.



(a) (b)

Figura 3.10: Exemplo de utilização com realidade aumentada

4 Validação

Após a implementação dos métodos descritos no Capítulo 3, é necessário verificar a qualidade da ferramenta criada. Como esta ferramenta é voltada para um público específico, foram realizados testes práticos com meteorologistas. Dois questionários foram criados para verificar a usabilidade do sistema, bem como o interesse dos especialistas em utilizá-lo.

O primeiro quesitonário, mais específico, contém 10 questões. Ele foi desenvolvido com base no questionário proposto por Brooke [5]. Neste texto, o autor defende que a usabilidade não é um conceito que pode ser definido globalmente. Ou seja, seu sentido é alterado dependendo do contexto onde está sendo utilizada. Assim, é impossível criar uma métrica para medir usabilidade independente do contexto do sistema sendo analisado. Usabilidade torna-se, então, a medida de quão *apropriado para certa tarefa* um sistema é.

No caso de sistemas computacionais, é necessário principalmente definir *quem* utilizará o programa, assim como *qual a finalidade* do uso desta ferramenta. Como definido no capítulo inicial, os usuários alvo desta ferramenta são meteorologistas que desejam realizar análises de cenários.

O primeiro questionário desenvolvido foi então aplicado a dois meteorologistas. Para avaliar a ferramenta, ambos especialistas utilizaram-na de modo a verificar todas suas funcionalidades, bem como extrair informações que achassem relevantes das visualizações geradas a partir do volume de dados meteorológicos. Após a utilização da ferramenta, os especialistas responderam o questionário e apresentaram sugestões das principais funções das quais sentiram falta.

Brooke também define uma métrica para avaliar os resultados da aplicação do questionário [5]. Ao final da aplicação do questionário temos um número entre 1 e 100 que informa o quão *usável* o sistema foi para aquele usuário. Os resultados obtidos com os especialistas foram satisfatórios, recebendo uma nota de 87,5 do primeiro usuário e 75 do segundo.

O segundo questionário foi apresentado para um grupo de 9 especialistas. Foi realizada uma apresentação onde a ferramenta foi demonstrada para eles. Ao final de apresentação um

questionário mais simples e genérico foi aplicado, para averiguar o interesse dos especialistas em utilizar uma ferramenta deste tipo. Foram apresentadas três questões objetivas sobre a usabilidade do programa e uma última questão que pedia ao questionado para descrever uma função que não foi implementada mas que ele acreditava ser importante para o usuário final.

Este segundo questionário também demonstrou um grande interesse da parte dos especialistas em utilizar esta ferramenta. A questão discurssiva mostrou-se muito importante para averiguar quais funcionalidades os meteorologistas gostariam de ver implementadas na ferramenta. Uma informação que eles gostariam de ver integradas com a ferramenta, por exemplo, é a de hidrografia. Atualmente, existe apenas uma informação da divisão geopolítica da região sendo estudada. Também foi sugerida a idéia de uma ferramenta de busca quando escolhendo qual modelo meteorológico abrir, de modo que o usuário possa buscar modelos por datas específicas.

A aplicação destes dois questionários demonstrou que o sistema, apesar de ainda não se encontrar numa fase operacional, já desperta grande interesse nos meteorologistas. Além disso, as ferramentas já integradas ao programa demonstraram-se de fácil uso, o que torna a exploração dos dados mais natural para os especialistas.

5 *Conclusão*

Foram apresentados os principais algoritmos e técnicas utilizadas para o desenvolvimento de uma aplicação de visualização interativa de dados meteorológicos. A parte de *integração* de todas estes diferentes métodos de visualização foi, com certeza, a parte mais trabalhosa do trabalho. A implementação de uma interface gráfica dinâmica foi o segundo maior esforço deste trabalho, apesar de não ser detalhada neste texto.

O cálculo de linhas de correntes e a realidade aumentada foram o foco principal no desenvolvimento desta aplicação, já que as outras modalidades de visualização apresentadas já estavam implementadas. O nível de interatividade alcançado pelo algoritmo de *streamlines* está dentro do desejado, tornando possível ao usuário visualizar em tempo real a descrição de um campo vetorial. É importante perceber, também, que a qualidade da visualização não ficou prejudicada pela interatividade do método.

O algoritmo de *Marching Cubes* desenvolvido pelo laboratório Cyclops foi mais uma vez testado, desta vez utilizando dados não médicos. A visualização gerada foi tão satisfatória quanto nas aplicações anteriores (considerando-se a pequena resolução do volume utilizado). Além disso, a aplicação de uma textura criada a partir de uma variável diferente da utilizada para a reconstrução da isosuperfície torna a visualização de dados mais rápida. O desempenho, no entanto, não foi tão bom quanto o esperado em volumes grandes, apesar de ainda ser interativo para os volumes testados.

A realidade aumentada, graças ao *framework* utilizado, mostrou-se estável e intuitiva. Com uma interface de simples uso, o ARToolKit é uma ótima escolha quando trabalhando em aplicações deste tipo. O reconhecimento dos padrões é preciso e, em suas últimas versões, a visualização se tornou bastante confiável.

Os questionários aplicados aos especialistas também mostraram que os usuários ficaram contentes com o resultado final da ferramenta, além de se mostrarem muito interessados em utilizá-la.

6 *Trabalhos Futuros*

Como visto nos capítulos anteriores, alguns pontos da implementação podem ser melhorados. O algoritmo de cálculo de *streamlines,* por exemplo, possui dois pontos principais a serem aprimorados.

Primeiramente, melhorar a escolha de sementes. Atualmente executamos um algoritmo aleatório de escolha de sementes, o que dá à visualização um fator aleatório. Duas execuções do algoritmo sobre um mesmo campo vetorial podem retornar linhas de correntes diferentes. Para eliminar este problema, é possível implementar um algoritmo de seleção de sementes mais inteligente, como o que utiliza uma triangulação de Delaunay, apresentado por [14]. Isto tornará a execução do algoritmo mais estável e confiável.

Outra melhoria possível no algoritmo é o cálculo de *streamlines* em três dimensões. Como os dados de vento e gradiente presentes nos arquivos GRIB utilizados em nossa aplicação apresentam dados tridimensionais, poderíamos construir linhas de corrente para representar todo este campo vetorial. Com três dimensões, porém, novos problemas precisam ser abordados. Em duas dimensões, por exemplo, não existe o problema de oclusão, ou seja, um conjunto de *streamlines* que esconde uma parte do campo vetorial. Em 3D é importante descobrir, além de por onde as linhas de corrente passam, quais delas são importantes para a visualização e quais podem ser ignoradas. Pontos críticos do volume devem ser visíveis mesmo quando o ângulo de visão do usuário não é ótimo.

Finalmente, uma pesquisa futura também poderia focar na paralelização de linhas de corrente. Isto visa aumentar o nível de interatividade do método. Com um volume muito grande (ou uma distância de separação muito pequena), o cálculo de *streamlines* pode se tornar pesado demais para suportar interatividade. Como a pesquisa se baseou principalmente em métodos não paralelos de cálculo, uma pesquisa mais voltada a este paradigma pode melhorar em muito o desempenho do algoritmo.

O algoritmo de *Marching Cubes* apresentado também não é paralelo, apesar de existirem artigos publicados que apresentam uma arquitetura paralela para esta técnica [9]. Esse aumento na

velocidade de reconstrução é fundamental, já que a aplicação, atualmente, deixa de ser interativa quando utilizando volumes grandes. Como boa parte da visualização já está sendo executada na GPU, uma pesquisa implementando um método paralelo de *marching cubes* utilizando a placa aceleradora gráfica pode aumentar em muito a interatividade da aplicação.

Atualmente, trabalhamos com três dimensões somente em nossa aplicação. É possível abrir volumes de diferentes datas, mas apenas isso. Seria interessante pesquisar a visualização de dados em quatro dimensões. Ou seja, tornar a visualização navegável também no eixo do tempo. Interpolando entre os diferentes volumes de dados que representam diferentes datas, podemos criar animações para mostrar aos especialistas a evolução do clima em uma região. Para tanto, é necessário o aumento da velocidade de cálculo das técnicas já utilizadas (apenas a visualização multi planar, por sua simplicidade, alcançaria velocidades interativas na implementação atual). Por isto, as pesquisas mencionadas anteriormente se tornam um requisito para esta.

Outro ponto pouco explorado neste trabalho é a visualização volumétrica. Com a reconstrução do algoritmo de *marching cubes*, possuímos uma isosuperfície que, apesar de estar no espaço tridimensional, mostra apenas onde no volume possuímos um mesmo valor. Podemos melhorar nossa visualização se fosse possível visualizar valores que estejam *dentro* de um intervalo de valores. Uma das técnicas que poderia ser utilizada para este fim é a de *Raycasting* [8].

Na parte de realidade aumentada, é seria interessante testar a implementação atual utilizando um *head-mounted display.* A versão atual utiliza a tela de um computador para mostrar o resultado para o usuário, o que faz com que ele veja ambas as cenas. Utilizando óculos especiais para realidade aumentada, a aplicação poderia ser testada em um ambiente ótimo. Além disso, o desenvolvimento de um *framework* próprio como base para a realidade aumentada pode retirar a dependência atual (a utilização do ARToolKit).

Finalmente, é necessária uma pesquisa de usabilidade da integração de todas as visualizações implementadas, bem como da interface gráfica. Testes com usuários leigos e especialistas podem ser efetuados para averiguar estes pontos, além de mostrar caminhos para melhoras.

Referências Bibliográficas

[1] Game: The eye of judgement - site oficial. Acessado em 28 de junho de 2011.

[2] Pagina oficial sobre as enchentes em santa catarina. Acessado em 28 de junho de 2011.

[3] *Marker tracking and HMD calibration for a video-based augmented reality conferencing system* (1999).

[4] AZUMA, R. A survey of augmented reality, 1997.

[5] BROOKE, J. *SUS: A quick and dirty usability scale*. Taylor and Francis, 1996, pp. 189–194.

[6] CHEN, Y., COHEN, J. D., AND KROLIK, J. H. Similarity-guided streamline placement with error evaluation.

[7] CORREA, A. G. D., DE ASSIS, G. A., NASCIMENTO, M. D., FICHEMAN, I., AND LOPES, R. D. D. Genvirtual: An augmented reality musical game for cognitive and motor rehabilitation. In *Virtual Rehabilitation, 2007* (sept. 2007), pp. 1 –6.

[8] ENGEL, K. *Real-time volume graphics*. Ak Peters Series. A K Peters, Ltd., 2006.

[9] GOETZ, F., JUNKLEWITZ, T., AND DOMIK, G. Real-time marching cubes on the vertex shader. *EUROGRAPHICS 2005 2005*, 1–4.

[10] JUAN, C., TOFFETTI, G., ABAD, F., AND CANO, J. Tangible cubes used as the user interface in an augmented reality game for edutainment. In *Advanced Learning Technologies (ICALT), 2010 IEEE 10th International Conference on* (july 2010), pp. 599 –603.

[11] JUAN, M., BOTELLA, C., ALCANIZ, M., BANOS, R., CARRION, C., MELERO, M., AND LOZANO, J. An augmented reality system for treating psychological disorders: application to phobia to cockroaches. In *Mixed and Augmented Reality, 2004. ISMAR 2004. Third IEEE and ACM International Symposium on* (nov. 2004), pp. 256 – 257.

[12] KITCHENHAM, B. Procedures for performing systematic reviews.

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

[14] MEBARKI, A., ALLIEZ, P., AND DEVILLERS, O. Farthest point seeding for efficient placement of streamlines.

[15] NEWMAN, T. S., AND YI, H. A survey of the marching cubes algorithm. *Computers & Graphics 30*, 5 (2006), 854 – 879.

[16] PAPATHOMAS, T. V., SCHIAVONE, J. A., AND JULESZ, B. Applications of computer graphics to the visualization of meteorological data. *SIGGRAPH Comput. Graph. 22* (June 1988), 327–334.

[17] PAYNE, B., AND TOGA, A. Surface mapping brain function on 3d models. *Computer Graphics and Applications, IEEE 10*, 5 (sep 1990), 33 –41.

[18] WANG, C.-Y., LEE, T.-F., AND FANG, C.-H. A volume visualization system with augmented reality interaction for evaluation of radiotherapy plans. In *Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on* (dec. 2009), pp. 433 –436.

[19] ZHANG, W., SUN, B., AND WANG, Y. A streamline placement method highlighting flow field topology.