VISUALIZAÇÃO DE ALTO DESEMPENHO DE DADOS OCEÂNICOS EM LARGA ESCALA

Andrei Gomes Lopes

Dissertação submetida ao corpo docente da Faculdade de Engenharia da Universidade do Estado do Rio de Janeiro – UERJ, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia da Computação.

Orientadora: Profª. Cristiana Barbosa Bentes.

Co-orientador: Prof. Ricardo Farias.

Programa de Pós-Graduação em Engenharia da Computação – Área de Concentração Geomática

Rio de Janeiro
abril/2007

LOPES, ANDREI GOMES

Visualização de Alto Desempenho de Dados oceânicos em Larga Escala [Rio de Janeiro] 2007.

viii, 61 p. 29,7 cm (FEN/UERJ, M.Sc., Programa de Pós-Graduação em Engenharia de Computação - Área de Concentração Geomática, 2007)

Dissertação - Universidade do Estado do Rio de Janeiro - UERJ

1. Computação Paralela 2. Visualização Volumétrica 3. Visualização de Oceanos

I. FEN/UERJ II. Título (série)

FOLHA DE JULGAMENTO

Título: Visualização de Alto Desempenho de Dados oceânicos em Larga Escala

Candidato: Andrei Gomes Lopes

Programa: Pós-Graduação em Engenharia de Computação – Área de Concentração Geomática

Data da defesa: 24 de abril de 2007

Aprovada por:

_____

Profª. Cristiana Barbosa Bentes, D.Sc., UERJ.

_____

Prof. Ricardo Farias, Ph.D., UFRJ.

_____

Prof. Marcelo Sperle Dias, D.Sc., UERJ.

_____

Profª. Maria Clícia Stelling de Castro, D.Sc., UERJ.

A

minha amada esposa Ana Cristina que sempre me apoiou incondicionalmente.

Agradeço aos Professores Cristiana Bentes e Ricardo Farias cuja orientação, apoio e incentivo foram fundamentais para realização deste trabalho.

Resumo da Dissertação apresentada à FEN/UERJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.).

VISUALIZAÇÃO DE ALTO DESEMPENHO DE DADOS OCEÂNICOS EM LARGA ESCALA

Andrei Gomes Lopes

abril/2007

Orientadora: Profª. Cristiana Barbosa Bentes, D.Sc., UERJ.
Co-orientador: Prof. Ricardo Farias, Ph.D., UFRJ.

Programa de Pós-Graduação em Engenharia de Computação – Área de Concentração Geomática

Os oceanos têm um papel fundamental na manutenção da vida na Terra. Recentemente, uma grande quantidade de dados de alta qualidade sobre vários aspectos dos oceanos tem sido gerada e disponibilizada na área de Oceanografia. O estudo e a interpretação destes dados requer o uso de ferramentas de visualização. O volume de dados oceânicos, entretanto, tem crescido consideravelmente ao longo dos anos e a geração de imagens para essa grande quantidade de dados tem alto custo computacional. Neste trabalho, estamos propondo uma ferramenta de visualização de oceanos de baixo custo, chamada OceanView 3D, que possa tratar eficientemente dados em larga escala e prover tempo de resposta interativo para dados menores. Nossa idéia é tratar o problema de custo computacional utilizando dois algoritmos diferentes de renderização volumétrica: um que explora a placa gráfica e outro que utiliza processamento paralelo para tirar proveito de uma arquitetura de baixo custo e alta disponibilidade como um cluster de PCs.

Estamos defendendo neste trabalho que nenhuma das duas formas de se resolver o problema de desempenho é ideal para todas as situações. Quando o dado é maior que a capacidade da memória da GPU, o uso de um cluster com renderização paralela é mais indicado, caso contrário, o uso da placa gráfica é mais eficiente. Com uma interface gráfica bastante amigável, realizamos alguns experimentos com dados provenientes do National Research Laboratory com resultados bastante satisfatórios.

Palavras-chave: computação paralela, visualização volumétrica, visualização de oceanos.

Abstract of Dissertation presented to FEN/UERJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

HIGH-PERFORMANCE ISSUES IN LARGE-SCALE OCEAN VISUALIZATION

Andrei Gomes Lopes

abril/2007

Advisors: Cristiana Barbosa Bentes and Ricardo Farias

After Graduation Program in Computer Engineering – Field of Geomatics

The oceans play an essential role in the maintenance of life on the planet. In order to better understand the characteristics of the oceans, the oceanographers strongly need a visualization tool. An image of the ocean conveys much more information than tables of numbers. The volume of ocean data has increased dramatically over the years and the generation of images of these huge datasets is computationally intensive. In this work, we are proposing a low-cost ocean visualization tool, called Oceanview 3D, that can efficiently to process data in a large scale and to supply interative answer time for smaller data. Our idea is to sort the computationally cost problem out using two different algoritmos of volumetric rendering: one that explores the graphic plate and another that uses parallel processing to benefit from a low-cost and high availability architecture as a PC cluster.

We are defending in this work that none of the two ways to solve the performance problem is ideal to all the situations. When the data is larger than the GPU memory capacity, the use of the cluster with parallel redering is the most indicated, otherwise, the use of the GPU is more efficient. With a very friendly graphic interface, we carried out some experiments with data that came from the National Research Laboratory with very succesful results.

Keywords: parallel computing, volume rendering, ocean visualization.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1
INTRODUÇÃO

Os oceanos Pacífico, Atlântico, Índico, Ártico, Antártico, juntamente com os mares interiores, formam uma grande massa de água que ocupa aproximadamente 71% da superfície de nosso planeta. Esta massa de água possui zonas abismais de mais de 11.000 metros de profundidade, como no Pacífico na fossa das Marianas [Wikipedia07]. Se considerarmos o volume, verificaremos que 97,4% de toda água de nosso planeta é salgada e que constitui uma enorme fonte para estudos dada a sua grande dimensão e importância para a vida na Terra.

As algas microscópicas que flutuam próximas a superfície dos oceanos produzem cerca de 90% da fotossíntese da Terra garantindo a existência de oxigênio em nossa atmosfera o que é fundamental para a existência de vida. Segundo teoria dominante, os oceanos foram o berço de origem da vida na Terra e até hoje tem papel fundamental em sua manutenção [Oparin38]. Sua importância é reconhecida, atualmente, de modo inequívoco em aspectos tão fundamentais e diversificados como o clima na Terra, o aproveitamento e exploração racional dos recursos biológicos e minerais, a detecção e o controle da poluição do mar.

Recentemente, uma grande quantidade de dados de alta qualidade sobre vários aspectos dos oceanos tem sido gerada e disponibilizada na área de Oceanografia. O estudo e a interpretação destes dados, entretanto, podem se tornar tarefas extremamente complicadas se o cientista não dispuser de ferramentas computacionais adequadas.

Dentre as ferramentas computacionais mais valiosas para os pesquisadores de oceanografia estão as de visualização volumétrica. A visualização volumétrica provê imagens tridimensionais dos oceanos, permitindo que os cientistas estudem o sistema natural com um grau de realismo sem precedentes, aumentando a capacidade de observação e relacionando as observações com os modelos. O estudo por meio da visualização volumétrica traz inestimáveis benefícios econômicos e sociais [Carvalho06].

O volume de dados que pode ser coletado por cientistas de oceanografia tem crescido consideravelmente na última década. As interações entre clima e oceano têm produzido enormes massas de dados que chegam a ordem de petabytes. Além disso, os dados oceânicos apresentam uma complexidade a mais que é sua natureza temporal [Ma00].

Portanto, o enorme crescimento na geração de dados oceânicos, juntamente com a sua dimensão temporal são o grande desafio para as ferramentas correntes de visualização de oceanos. O tratamento de grandes massas de dados gera uma grande demanda de memória e computação no processo de visualização.

As ferramentas de visualização existentes não estão preparadas para tratar eficientemente essa quantidade de dados [Gary01, Liu05]. Principalmente quando se deseja visualizar a estrutura interna dos oceanos e, portanto, utilizar técnicas de renderização volumétrica. A renderização volumétrica é computacionalmente muito custosa. Quando a quantidade de dados a serem renderizados é muito grande, mesmo o processador mais veloz atualmente não tem capacidade de executar tantos cálculos em uma pequena quantidade de tempo e prover tempo de resposta interativo para o usuário.

Para tratar o enorme crescimento dos dados oceânicos, usualmente as ferramentas de visualização distribuem o processamento por diferentes computadores, implementando algoritmos paralelos de renderização. Esta solução promove a aceleração da computação e permite a distribuição dos dados. Entretanto, inclui o overhead de comunicação entre os processadores.

Uma outra solução que tem sido adotada para o problema do custo computacional da renderização é o uso de placas gráficas programáveis. Placas gráficas implementam as operações gráficas em hardware obtendo ganhos significativos de desempenho. Atualmente, tem sido a melhor solução para a geração de imagens em tempo-real, porque provê taxas de dezenas de imagens geradas por segundo. Esta solução, entretanto, não permite tratar dados muito grandes devido à limitação de memória das placas gráficas atuais.

Como as ferramentas de visualização optam por utilizar apenas um algoritmo de renderização, o usuário deve optar entre ter imagens em tempo-real ou tratar dados muito grandes.

Neste trabalho, é proposta uma ferramenta de visualização de oceanos de baixo custo, chamada OceanView 3D, que possa tratar eficientemente dados em larga escala e prover tempo de resposta interativo para dados menores. A idéia é tratar o problema de custo computacional utilizando dois algoritmos diferentes de renderização volumétrica: um que explora a placa gráfica e outro que utiliza processamento paralelo para tirar proveito de uma arquitetura de baixo custo e alta disponibilidade como um cluster de PCs. Defende-se neste trabalho que nenhuma das duas formas de se resolver o problema de desempenho é ideal para todas as situações. Quando o dado é maior que a capacidade da memória da GPU, o uso de um cluster com renderização paralela é mais indicado, caso contrário, o uso da placa gráfica é mais eficiente.

O sistema OceanView 3D foi concebido para ser uma ferramenta de fácil uso e que reúne numa única aplicação a possibilidade de explorar diferentes recursos de renderização volumétrica em alto desempenho. Os algoritmos de renderização de alto desempenho foram avaliados separadamente e ambos atingiram resultados satisfatórios. Pretendemos assim fornecer aos especialistas em oceanos a capacidade de converter dados numéricos em imagens, permitindo a extração direta de informações sobre fenômenos do oceano de difícil observação, contribuindo sensivelmente em áreas como: gerenciamento de pesca e de mamíferos, mineração e indústria petrolífera, e pesquisas sobre o clima e meio ambiente.

1.1 Objetivos

Os principais objetivos desse trabalho são:

- Proposta de uma ferramenta computacional para visualização de dados oceânicos que integre os conceitos de visualização volumétrica e de mecanismos de alto desempenho computacional com baixo custo.

- Prover uma ferramenta que seja capaz de converter formatos de dados oceânicos a fim de serem utilizados, pois não há padronização de formatos.

- Avaliação do desempenho dos mecanismos para obtenção de imagens tridimensionais de dados oceânicos. Resultados experimentais mostram que os mecanismos computacionais adotados apresentam eficiência satisfatória quando executados numa estação com placa de vídeo gráfica com dados que cabem em memória e num cluster de 16 processadores para dados de maior dimensão que não cabem em memória.

1.2 Organização da Dissertação

Este trabalho está organizado da forma descrita a seguir. No Capítulo 2, discute-se os conceitos básicos relacionados às áreas de domínio do sistema OceanView 3D: Visualização Científica e Dados oceânicos. O Capítulo 3 aborda questões sobre Visualização de Alto Desempenho de Oceanos. O Capítulo 4 detalha a ferramenta desenvolvida para visualização de dados oceânicos – OceanView 3D. No Capítulo 5 são mostrados os resultados experimentais. No Capítulo 6 exibe-se as conclusões e sugestões para trabalhos futuros.

CAPÍTULO 2
CONCEITOS BÁSICOS

Este trabalho integra conceitos de domínios diferentes, como: Visualização Científica, Dados oceânicos e Computação de Alto Desempenho. Portanto, este capítulo pretende apresentar um resumo sobre alguns conceitos básicos importantes para as áreas de conhecimento envolvidas.

2.1. Visualização Científica

A Visualização Científica é uma área de pesquisa que evolui bastante nos últimos anos e estuda estratégias e algoritmos para mapear informações, em geral numéricas, em imagens, com o auxílio de técnicas de Computação Gráfica [Paiva99]. A Visualização Científica constitui-se como uma poderosa ferramenta para a interpretação e análise de dados computacionais de caráter científico, que representam fenômenos complexos.

Sabe-se que o cérebro humano possui grande habilidade de estabelecer e reconhecer padrões visuais a partir de imagens. Por outro lado, possui dificuldade de estabelecer e reconhecer padrões numéricos associados a uma tabela com grande número de elementos, por exemplo. O crescimento expressivo da quantidade de dados produzida por computadores em simulações ou por sensores, como os embarcados em navios, satélites ou radiotelescópios, torna inviável a interpretação e análise dos dados brutos (por exemplo, na forma de tabelas). O foco da Visualização Científica é, portanto, o de converter dados numéricos em imagens de forma a permitir que especialistas possam extrair de forma fácil e direta informações substanciais sobre um fenômeno do qual se possui apenas dados numéricos.

Diversas áreas de conhecimento, como Medicina (em ressonância magnética, tomografia computadorizada e simulação de modelos cardiovasculares), Física (em simulação de modelos subatômicos), Biologia (em bioinformática e biologia molecular computacional) e Engenharia (em modelagem de componentes mecânicos e simulações de dinâmicas de fluidos), têm se beneficiado de ferramentas de visualização volumétrica. Na área de Oceanografia, a Visualização Científica provê imagens tridimensionais dos oceanos, permitindo que os cientistas estudem o sistema natural com um grau de realismo sem precedentes.

Dados a serem visualizados podem ser obtidos de diferentes formas. Através de dispositivos de captura como tomografia computadorizada ou ressonância magnética; simulações (elementos finitos, diferenças finitas); satélites de observação da Terra;

levantamentos geofísicos; bóias no oceano; ou técnicas de modelagem. Estes dados compreendem um conjunto de pontos no espaço tridimensional, onde a cada ponto estão associados propriedades ou fenômenos observados. São chamados dados volumétricos. Os valores nas posições intermediárias, podem ser obtidos pela interpolação entre os valores periféricos. No sistema apresentado neste trabalho, utiliza-se a interpolação linear. Segundo Kaufman [Kaufman88], dados volumétricos são tipicamente um conjunto *S* de amostras *(x, y, z, v)*, sendo *v* o valor de alguma propriedade mensurável dos dados, como cor e densidade, e *(x, y, z)* a sua localização no espaço tridimensional. Por exemplo, no caso de dados coletados do oceano, essas propriedades podem ser sua temperatura, densidade ou salinidade.

Os dados volumétricos são representados como uma estrutura geométrica em forma de malha poliedral composta por células tridimensionais chamadas voxels ou células. A malha é considerada regular se seus voxels possuem formas cubos idênticos, vide a Figura 2.1 que exemplifica uma típica malha regular. Caso a malha seja composta por voxels poliedrais diferentes entre si, ela é considerada como sendo uma malha irregular. Na Figura 2.2 é demonstrada a representação de células de malhas irregulares.

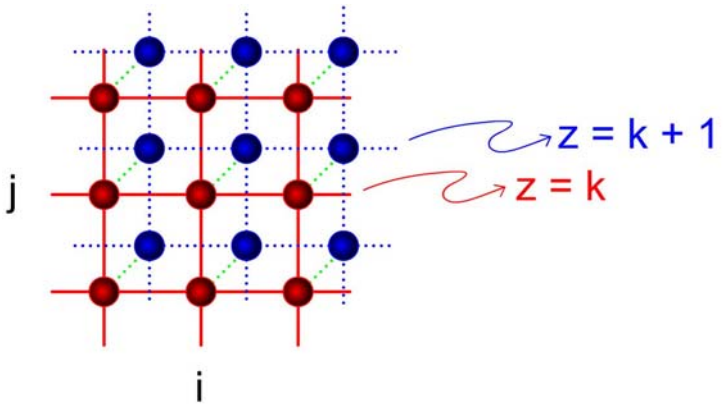

Figura 2.1: Representação implícita de malhas regulares

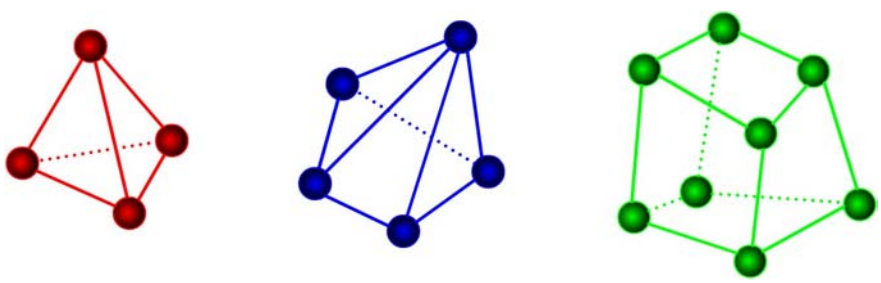

Figura 2.2: Representação de células de malhas irregulares.

As técnicas de visualização podem ser classificadas em dois tipos: ***surface rendering*** (ou "visualização através de superfícies") e ***volume rendering*** (ou "visualização direta de volume")[Kaufman88]. A visualização pode retratar somente a superfície do corpo por meio de uma renderização de superfície, conforme ilustrado na Figura 2.3, ou retratar todo o interior do corpo por meio de uma renderização de volume, conforme ilustrado na Figura 2.4.

Técnicas de visualização através de superfícies envolvem a extração e a representação de uma isosuperfície que é posteriormente visualizada. Neste caso, apenas os dados da superfície do volume são representados na imagem. As grandes vantagens desta técnica, entretanto, são a velocidade para a geração e exibição da imagem final e o pouco espaço de armazenamento requerido. Representações deste tipo são apropriadas quando existem isosuperfícies bem definidas nos dados, mas não são eficientes quando o volume é composto por muitas microestruturas, tais como os tecidos em muitas imagens médicas [Mueller99].

A segunda classe, visualização direta de volume, também chamada de renderização de volume, consiste em representar o volume através de voxels 3D que são projetados diretamente em pixels 2D e armazenados como uma imagem. O seu principal objetivo é exibir as estruturas existentes no interior do volume de dados, permitindo a identificação e exploração de estruturas internas no volume.

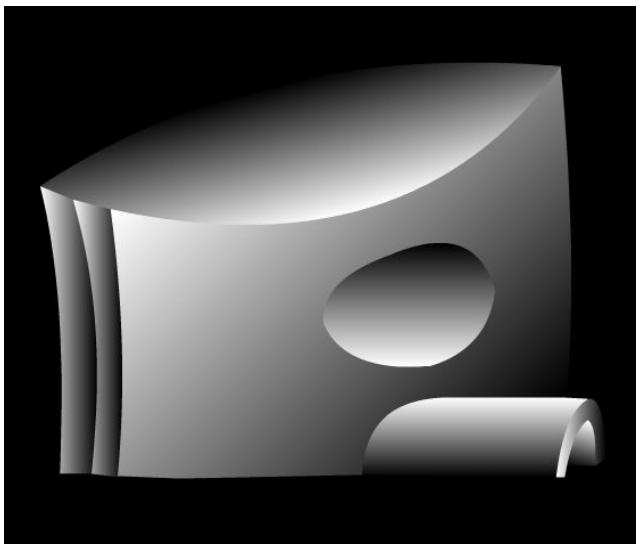

Figura 2.3: Visualização com renderização de superfície de um objeto tridimensional real.

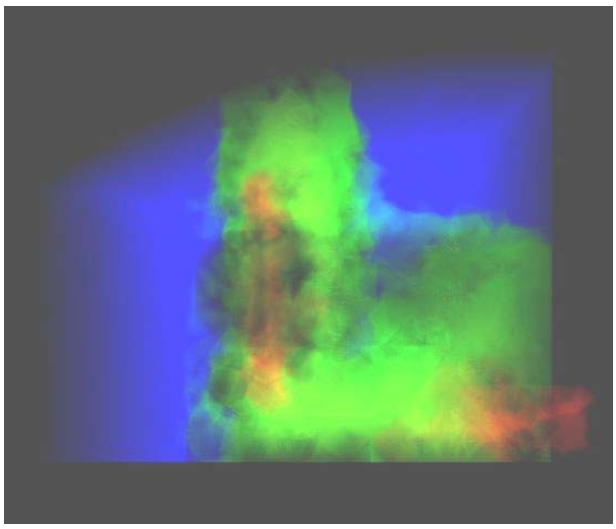

Figura 2.4: Visualização com renderização de volume de um objeto tridimensional real.

2.2. Renderização de Volume

A renderização de volume exibe a massa de dados como um corpo translúcido, de forma que possamos visualizar todo o seu interior e não apenas sua superfície. Podemos interpretar o processo como a obtenção de uma imagem de raio X do objeto. No caso de visualização de dados oceânicos, a renderização de volume se mostra mais eficiente, porque estes dados estão intimamente ligados ao interior do volume retratado, conforme mostra a Figura 2.5. Nesta Figura podemos observar duas imagens de oceanos, a Figura 2.5(a) mostra apenas a superfície do oceano e a Figura 2.5(b) mostra tanto dados da superfície como do interior. Nesta última podemos observar que determinados pontos do oceano possuem alta temperatura na superfície, mas não apresentam a mesma temperatura em camadas inferiores. Visualizar a estrutura interna dos oceanos permite uma exploração mais detalhada de características como temperatura, velocidade, salinidade ou densidade, permitindo também uma análise dinâmica da variação destas características ao longo do tempo.

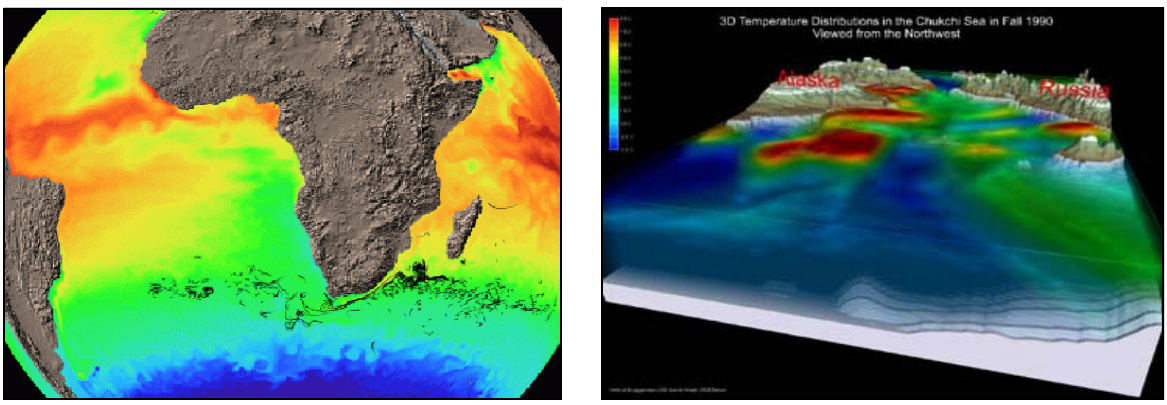

Figura 2.5: (a) Visualização da temperatura do oceano somente da sua superfície; (b) Visualização da temperatura do oceano com detalhes do seu interior.

Dentre os algoritmos de renderização de volume existentes, podemos destacar os baseados na técnica de *ray-casting* e no paradigma de *sweep plane.*

2.2.1. Técnica de *ray-casting*

Na técnica de *ray-casting*, para cada pixel do plano da imagem gerada é lançado um raio que percorre o conteúdo do corpo no ângulo de observação. Cada voxel cortado pelo raio contribui com cor e opacidade e o somatório das contribuições resulta na cor final do pixel. Na Figura 2.6, ilustramos o princípio de funcionamento da técnica de *ray-casting*. É apresentado na Figura um raio correspondente a um determinado pixel da tela, cortando apenas um voxel do volume. Os dois pontos de interseção do raio no voxel, z_1 e z_2, são utilizados para se calcular o quanto esta fatia do volume interfere na cor e na opacidade do pixel em questão. Para cada voxel intersectado pelo raio a mesma computação é realizada, até que o raio deixe o volume.

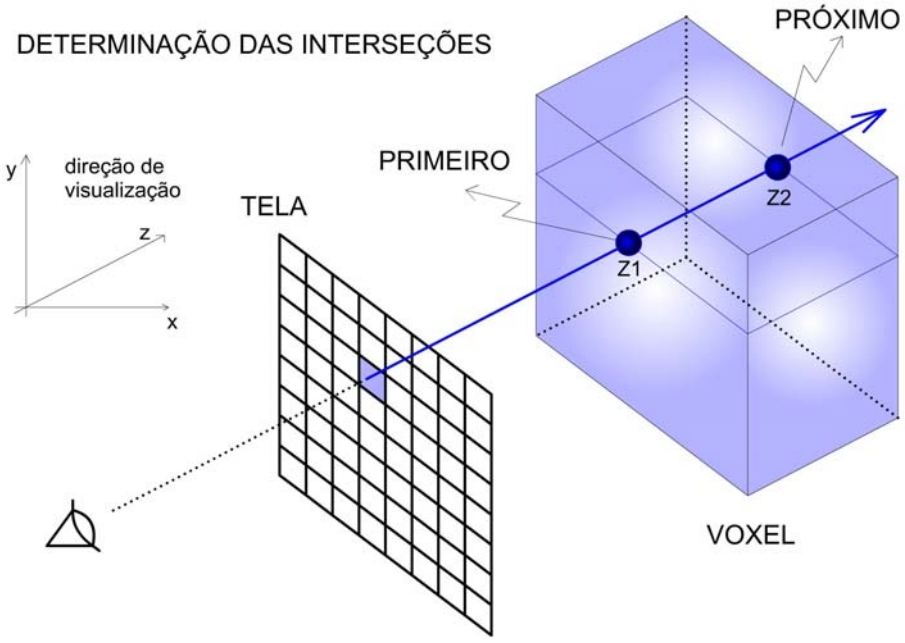

Figura 2.6: Representação da técnica de *ray-casting*

2.2.2. Paradigma de *sweep plane*

No paradigma de *sweep plane*, um plano paralelo ao plano da imagem percorre o corpo na direção do eixo z e a medida que ele transpassa o corpo, as faces dos voxels são projetadas na tela formando a imagem. Na Figura 2.7 ilustramos o processo para melhor entendimento. Cada face projetada na tela gera um conjunto de interseções com uma série de pixels. Essas interseções são utilizadas, tal qual no algoritmo de *ray-casting*, para o cálculo da contribuição da face na cor e opacidade final de cada pixel.

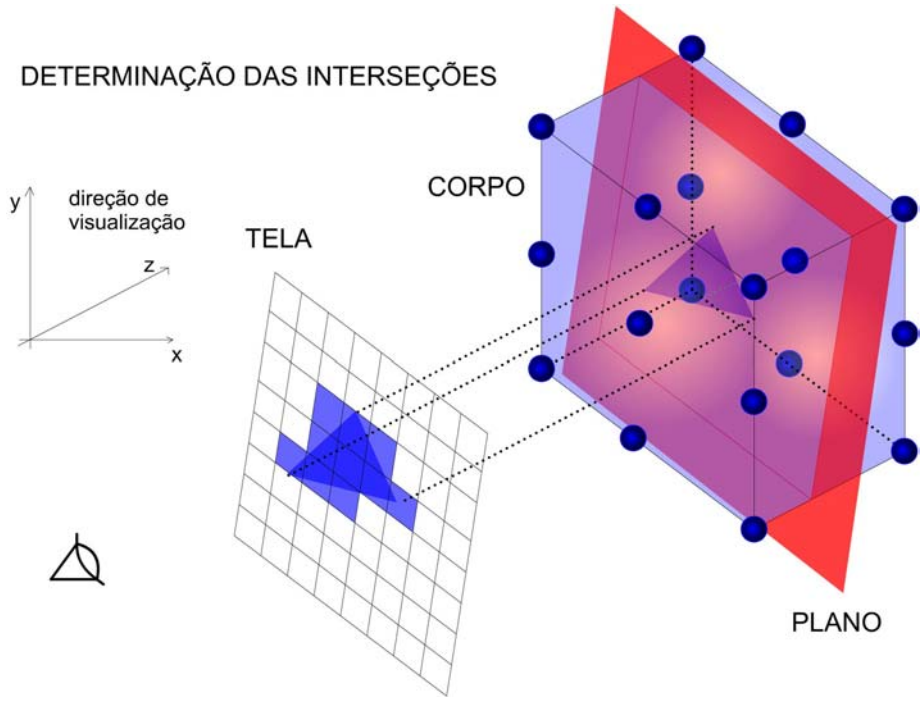

Figura 2.7: Representação do paradigma de *sweep plane*

Neste trabalho estamos particularmente interessados no algoritmo de renderização baseado em *sweep plane*, chamado Zsweep, desenvolvido pelo Prof. Ricardo Farias da COPPE/UFRJ. Este algoritmo provou ser um algoritmo de visualização volumétrica veloz, apresentando uso eficiente de memória [Farias00]. Além disso, ele possui versão paralela que pode de ser executada em cluster de PC's e permite o processamento de dados *out-of-core*, ou seja, os dados podem ter tamanhos superiores a capacidade de memória do computador. Estas características o tornam apropriado para o processamento de dados oceânicos, cujo tamanho pode chegar ordem de terabytes.

2.3. Computação de Alto Desempenho

Independente do algoritmo de renderização utilizado, a visualização volumétrica de grandes massas de dados é um problema conhecidamente dispendioso em termos computacionais. Mesmo os algoritmos mais velozes gastam um tempo de execução substancial para executar em um único processador. A solução para o problema do custo computacional da renderização de volume tem seguido, atualmente, por dois caminhos distintos: uso de hardware gráfico e computação paralela. O uso da placa gráfica para acelerar as operações de renderização tem levado a grandes avanços na área de visualização e obtido resultados promissores. Esta solução, entretanto, está altamente ligada à disponibilidade tecnológica e aos avanços na área de hardware. O uso da computação paralela é uma alternativa mais genérica e permite soluções de baixo custo.

2.3.1. Placas Gráficas

O poder de processamento do hardware especializado para as operações de computação gráfica, também chamado de GPU (Graphics Processing Unit), tem crescido num ritmo superior ao dos processadores convencionais. Este avanço se deve em grande parte ao apelo comercial dos jogos eletrônicos. A indústria de jogos eletrônicos fatura mais que a indústria de cinema atualmente [ABRAGAMES04].

Apesar do seu forte apelo comercial, a utilização de placas gráficas para a aceleração da visualização volumétrica tem sido foco de diversas pesquisas [Maximo06, Combo03]. O trabalho apresentado em [Comba03] divide a evolução das placas gráficas em quatro gerações diferentes. A primeira geração apareceu em torno de 1998 com placas NVidia e ATI, que operavam apenas na porção de rasterização do pipeline de renderização. A segunda geração apareceu no fim de 1999 e início do ano 2000, tinha mais capacidade de processamento do pipeline de renderização além da rasterização, mas ainda não tinha capacidade de programação. A terceira geração de placas gráficas surgiu em 2001 e 2002 com as placas NVidia GeForce3 e GeForce4, e a ATI Radeon 8500. O grande avanço destas placas foi a sua capacidade de programação, embora o programa fosse limitado a 128 instruções, sem a possibilidade de se utilizar instruções de desvio. A quarta geração compreende o estado-da-arte de placas gráficas que são programáveis, cujos programas podem ter mais de 1000 instruções, incluindo instruções de desvio. Elas são otimizadas para o processamento de dados vetoriais, permitindo executar simultaneamente uma operação para cada uma das componentes de um vetor de dimensão 4. Diversas operações aritméticas (p. ex., seno,

exponencial, produto interno) comumente utilizadas são eficientemente implementadas com apenas uma instrução em linguagem de máquina.

O aumento da capacidade e da programabilidade das placas gráficas tem trazido grandes benefícios em termos de desempenho para implementação de algoritmos de visualização. A GPU é utilizada como um *stream processor* e permite a geração de imagens em tempo-real para massas de dados de tamanho limitado. Os dados enviados para a placa gráfica são os vértices de primitivas poligonais (geralmente pontos, linhas ou triângulos) e alguns atributos adicionais, como cor e coordenadas de textura. Os vértices são individualmente processados, na etapa de *geometria*, e enviados para a *rasterização*, que é responsável por preencher cada posição (*pixel*) da tela de projeção pertencente à primitiva.

2.3.2. Computação Paralela

O processamento paralelo e distribuído tem sido largamente empregado em sistemas de computação com o objetivo de aumentar o desempenho de problemas de grande escala. A idéia é reduzir tempo para gerar informações realizando tarefas simultaneamente. Aplicações de visualização científica utilizam o processamento paralelo através da paralelização dos algoritmos de visualização e de sua implementação em arquiteturas com múltiplos processadores.

Atualmente, a computação paralela pode ser realizada em diversos ambientes. A visualização paralela tem sido foco de uma série de estudos [Challinger93, Hofsetz00, Ma95, Ma97, Ma01]. Algoritmos paralelos foram propostos, produzindo melhoras significativas de desempenho em máquinas como SGI Power Challenge, IBM SP2 ou SGI Origin 2000. Essas máquinas paralelas, no entanto, são extremamente caras. Entretanto, um dos motivos determinantes para a escolha da arquitetura paralela é o custo.

Recentemente, o baixo custo e a alta disponibilidade de computadores pessoais (PCs) e a tecnologia de redes, tornou um *cluster* de PCs uma arquitetura atraente para a execução paralela de aplicações de grande escala, como a visualização de dados volumétricos [Schneider98]. Um *cluster* pode ser definido como um conjunto de nós processadores (PCs ou estações) autônomos e que interligados comportam-se como um sistema de imagem única e que atuam de forma cooperativa com objetivo de diminuir o tempo de processamento de uma determinada tarefa.. O conceito de imagem única dita que um sistema paralelo ou distribuído, independente de ser composto por vários processadores ou recursos geograficamente distribuídos, deve comportar-se com um sistema centralizado do ponto de vista do usuário. Dessa forma, todos os aspectos relativos à distribuição de dados e de tarefas,

comunicação e sincronização entre tarefas e a organização física do sistema devem ser abstraídos do usuário, ou seja, devem ser transparentes a ele. Um *cluster* de PCs pode ser facilmente atualizado com as novas tendências de tecnologia. Além disso, quando as necessidades de computação se estendem para além dos recursos de um *cluster*, podemos empregar ambientes heterogêneos, como, por exemplo, ambientes de *grid* [Cirne01, Foster03, Foster98].

Os ambientes de *grid*, também chamados de ambientes de computação oportunista, fornecem a possibilidade de se utilizar a base instalada de computadores pessoais de maneira a realizar computação utilizando a parte dos recursos que, de outra forma, estariam ociosos. Ambientes de *grid* têm sido temas de pesquisas bastante recente em sistemas distribuídos. Isso porque exploram a potencialidade de se alocar uma enorme quantidade de recursos (como centenas de milhares de computadores conectados via Internet) a uma aplicação paralela. Devido ao grande potencial da área e ao interesse que vem recebendo na comunidade científica nacional e internacional, é interessante investigar, também, essa arquitetura como um ambiente de execução para a visualização paralela de oceanos.

2.4. Dados Oceânicos

Os oceanos são alvos de estudos de diversas organizações mundiais e grandes quantidades de informações são colhidas pelos mais diversos tipos de sensores e equipamentos. Hoje podemos obter conjuntos de dados de um trecho do oceano que retratam salinidade, velocidade escalar, temperatura, pressão, densidade, concentração de oxigênio, etc. Todas estas grandezas estão associadas a localizações espaciais tridimensionais. Podemos caracterizar o georeferenciamento destes dados pelas coordenadas de latitude, longitude e profundidade, caracterizando assim os 3 eixos espaciais. Na verdade, dados coletados de um oceano compreendem a um conjunto de pontos no espaço tridimensional, onde a cada ponto estão associados propriedades ou fenômenos observados.

Infelizmente, não há uma padronização nos dados obtidos pelas organizações, cada uma possui uma formatação própria e conseqüentemente isto dificulta muito a representação e análise dos mesmos. Quando estamos interessados em analisar os dados de oecano através de imagens tridimensionais, eles devem ser representados por uma estrutura geométrica renderizável. Embora a transformação de um conjunto de pontos no espaço em uma malha tridimensional, não seja uma tarefa complicada (basta conectar os pontos em poliedros e armazenar a estrutura de vizinhança dos poliedros), ela é totalmente dependente do formato em que o dado foi gerado, pois se os dados coletados estiverem muito esparsos, ou os valores coletados não possuírem boa precisão, a imagem gerada pode não retratar de forma adequada o fenômeno.

2.5. Visualização de Oceanos

Visualizar a estrutura interna dos oceanos permite uma exploração mais dinâmica e detalhada de características como temperatura, velocidade, salinidade ou densidade. Além disso, permite a análise de fenômenos que não poderiam ser observados somente com a visualização de superfície. Como por exemplo, o fluxo de saída do Mediterrâneo. No mar Mediterrâneo, a evaporação produz uma água mais densa e salgada do que a água do oceano Atlântico. Portanto, a água passa através do estreito de Gibraltar, formando uma massa de água distinta. Este fenômeno só pode ser completamente analisado visualmente com técnicas de renderização volumétrica, que permitem mostrar as diferenças de massas no interior dos mares. A visualização do interior dos oceanos e mares pode ter contribuições valiosas para diversas áreas de aplicação, como por exemplo:

- **Gerenciamento de Pesca e de Mamíferos:** Ante a extensão e riqueza do ambiente marinho podemos imaginar o potencial de pescado que ele fornece ao homem. Calcula-se hoje em dia que a pesca atinge cerca de 90 milhões de toneladas ao ano. A pesca tem um significado protéico e econômico muito grande para o ser humano, pois representa fonte de alimento e de renda. Técnicas de renderização de volume podem ser usadas para determinar propriedades do oceano que podem explicar o comportamento dos peixes e mamíferos, ajudando a pesquisa sobre recursos e habitat oceânicos.

- **Mineração e Indústria Petrolífera:** Os oceanos fornecem outras riquezas econômicas importantíssimas como : o petróleo, minerais como ouro, prata, cobre, ferro, zinco, etc. O petróleo juntamente com o pescado formam talvez as duas mais importantes riquezas do

mar na atualidade, mas há outras que podem ser exploradas em grande escala, como as já citadas. A mineração nas profundezas dos oceanos, que pode se constituir no próximo século como uma grande fonte econômica já está em franco crescimento e logo atingirá foro mundial ante o desenvolvimento da tecnologia que vem permitindo pesquisar em águas profundas. A visualização das correntes oceânicas ajuda na colocação de dutos, equipamentos e plataformas em águas de grande profundidade, evitando e minimizando os impactos que essas correntes podem causar.

- **Pesquisas sobre o Clima:** Os oceanos atuam como regulador do clima, pois as correntes marítimas levam água quente dos trópicos aos pólos influenciando no clima global [Rosa04]. Os estudos sobre a circulação geral dos oceanos têm importância fundamental no diagnóstico e previsão climática. A visualização e modelagem das mudanças na distribuição de calor nos oceanos permitem aos cientistas preverem mudanças climáticas e seus impactos nas diversas atividades humanas.

CAPÍTULO 3
VISUALIZAÇÃO DE ALTO DESEMPENHO DE OCEANOS

As técnicas de visualização para dados oceânicos devem prover resultado em tempo interativo de modo a melhorar a qualidade do entendimento do comportamento e características dos oceanos. O desenvolvimento de uma ferramenta interativa, que provê renderização de volume para dados oceânicos de larga escala, não é uma tarefa simples. Portanto, nosso foco aqui é mostrar duas soluções de alto desempenho diferentes para acelerar o processo de renderização: explorar as facilidades de programação de placas gráficas e explorar o paralelismo potencial de um cluster de PCs.

3.1. Explorando o Hardware Gráfico

Apesar das limitações de programação de uma GPU, a grande velocidade atingida nas operações gráficas as tornam uma excelente alternativa para aceleração da renderização volumétrica.

Neste trabalho, utilizamos um algoritmo de renderização volumétrica escrito para placa gráfica descrito em [Máximo06]. Este algoritmo é dividido em três grandes partes, uma executa na CPU e as outras duas executam na GPU. No primeiro passo, a CPU envia para a placa gráfica as informações dos vértices e das células. O primeiro programa da GPU é chamado e computa para todos os tetraedros, o tipo de projeção e a coordenada z do centróide de cada tetraedro. A lista de centróides é enviada para a CPU. No segundo passo, a CPU ordena a lista de centróides, para definir a ordem de projeção. O terceiro passo é realizado pelo segundo programa da GPU, que interpola os valores escalares dos vértices para computar os valores de cor e opacidade na imagem final. Ilustramos o processo na Figura 3.1.

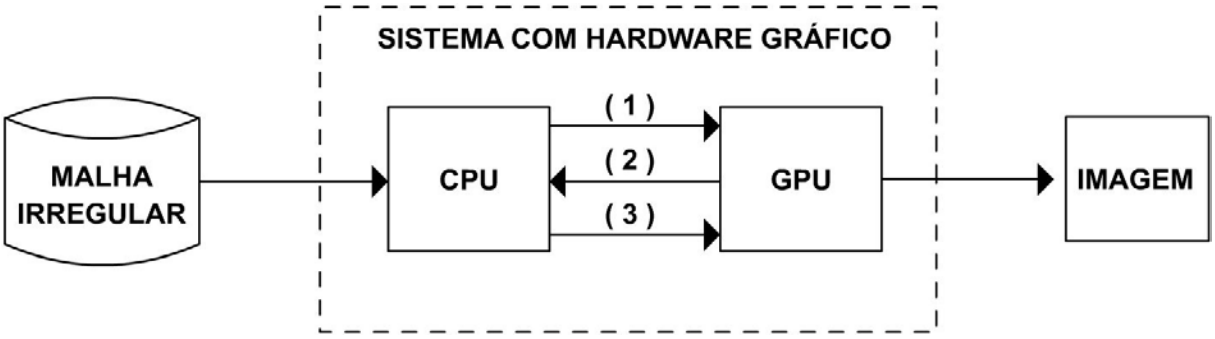

Figura. 3.1: Diagrama da renderização no sistema com hardware gráfico: (1) O programa da GPU computa para todos os tetraedros, o tipo de projeção e a coordenada z do centróide de cada tetraedro; (2) A lista de centróides é enviada para a CPU onde é ordenada para projeção; (3) O programa da GPU interpola os valores escalares dos vértices para computar os valores de cor e opacidade na imagem final.

Para acelerar a comunicação entre a CPU e a placa gráfica, o algoritmo utiliza arrays de vértices e as primitivas são desenhadas como triângulos utilizando comandos especiais de OpenGL (Open Graphic Language), isto é demonstrado na Figura 3.2 onde os tetraedros são rebatidos num plano e convertidos em triângulos. Estes comandos nos permitem enviar os dados para a placa gráfica como um grande pacote, evitando assim o gargalo causado pela baixa banda passante do barramento da placa gráfica. Assim que o dado é armazenado na memória da placa gráfica, o processamento pode iniciar.

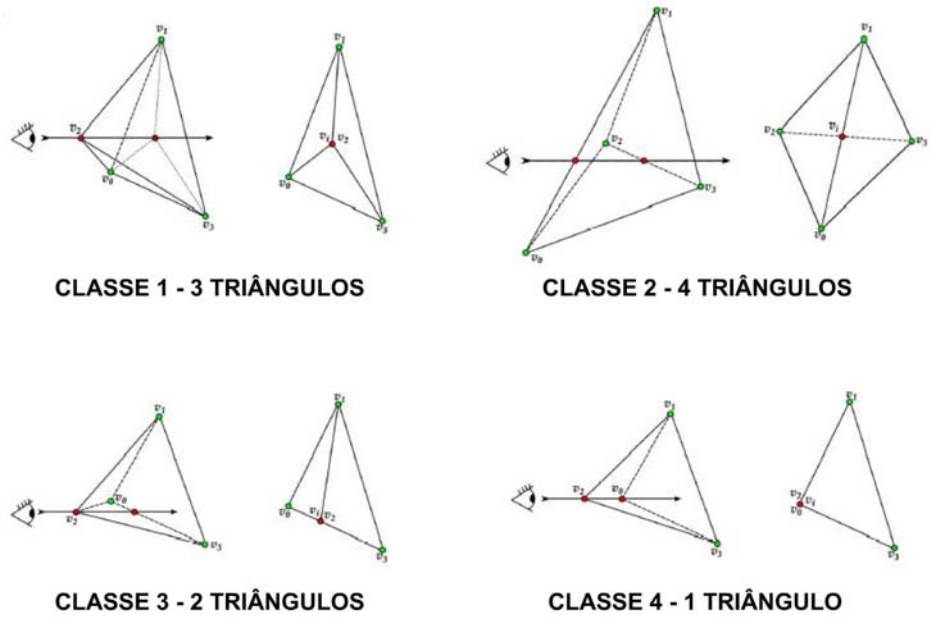

Figura 3.2: Conversão de tetraedros em triângulos de acordo com o ângulo de visão

3.2. Explorando a Renderização Paralela

O sistema paralelo de renderização que utilizamos neste trabalho é chamado DPZSweep [Coelho04], que é uma versão distribuída do sistema ZSweep que executa em um cluster de PCs. Neste sistema, cada nó de processamento do cluster realiza a renderização de uma porção da imagem e ao final do processamento, as porções geradas são reunidas de modo a gerar a imagem final.

O algoritmo paralelo utilizado usa uma divisão de tarefas no espaço da imagem. A tela é dividida em retângulos chamados *tiles*, conforme mostra a Figura 3.2. Cada *tile* representa uma unidade computacional independente. Pela identificação do *tile*, o processador sabe qual parte do dado ele deve renderizar. A renderização de um *tile* é feita por meio do algoritmo sequencial ZSweep [Farias00].

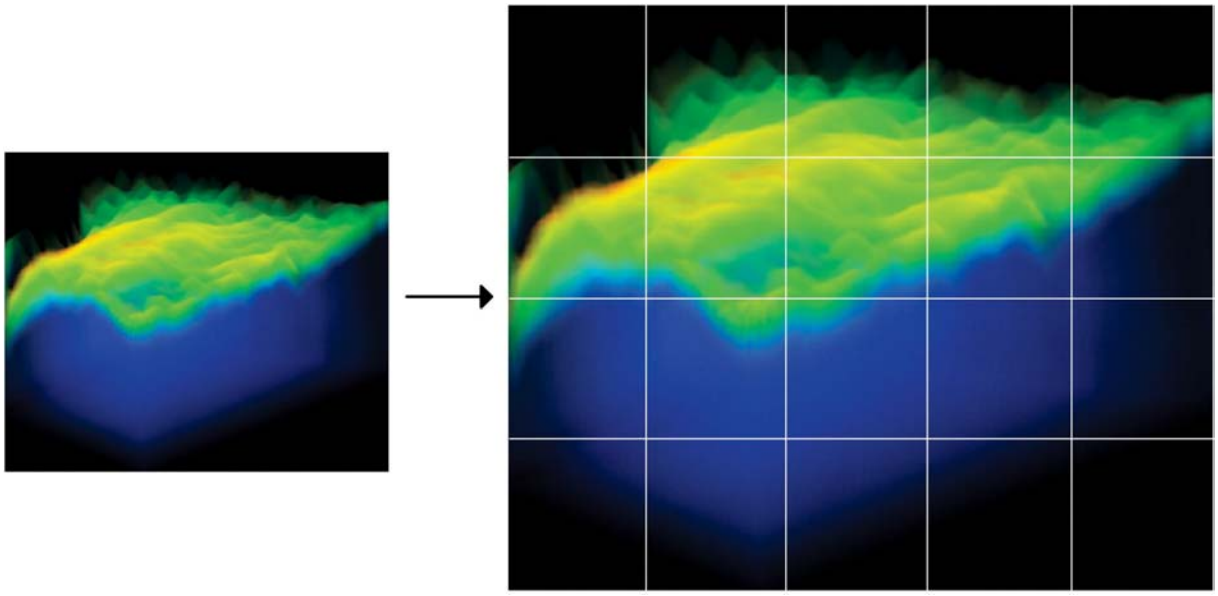

Figura 3.2: Divisão da imagem em *tiles*

O sistema DPZSweep trata de dois problemas fundamentais na renderização paralela em cluster de PCs:

- Balanceamento de carga – dado que não se pode determinar de antemão o custo do *rendering* de uma região do volume, para resolver este problema o sistema DPZSweep utiliza algoritmos próprios para balanceamento dinâmico da carga entre os diversos elementos de processamento do cluster;

- Uso da memória – devido ao tamanho limitado das memórias de cada PC de um cluster, muitas vezes o conjunto de dados a ser renderizado não cabe na memória principal, para resolver este problema o sistema DPZSweep utiliza algoritmo *out-of-core* que mantém a massa de dados em disco e carrega na memória, por demanda, apenas a porção dos dados que será renderizada de cada vez.

Na Figura 3.3, apresentamos um diagrama simplificado do sistema de renderização paralela. Conforme podemos observar na Figura 3.3, os dados volumétricos estão representados por uma malha irregular de voxels. Esta malha contém o conjunto de dados de entrada a serem renderizados, mas não pode ser carregada inteira na memória principal. O conjunto de dados de entrada é, então, particionado numa etapa de pré-processamento para produzir uma representação hierárquica dos dados numa estrutura denominada *octree*. O algoritmo paralelo então distribui o trabalho a ser realizado pelos elementos de processamento do cluster e cada elemento de processamento lê do disco as porções da malha que irá renderizar de acordo com a estrutura da *octree*. A renderização realizada por cada elemento de processamento é independente e a porção da imagem computada por cada um deles é retornada a um único elemento, responsável por exibir a imagem final.

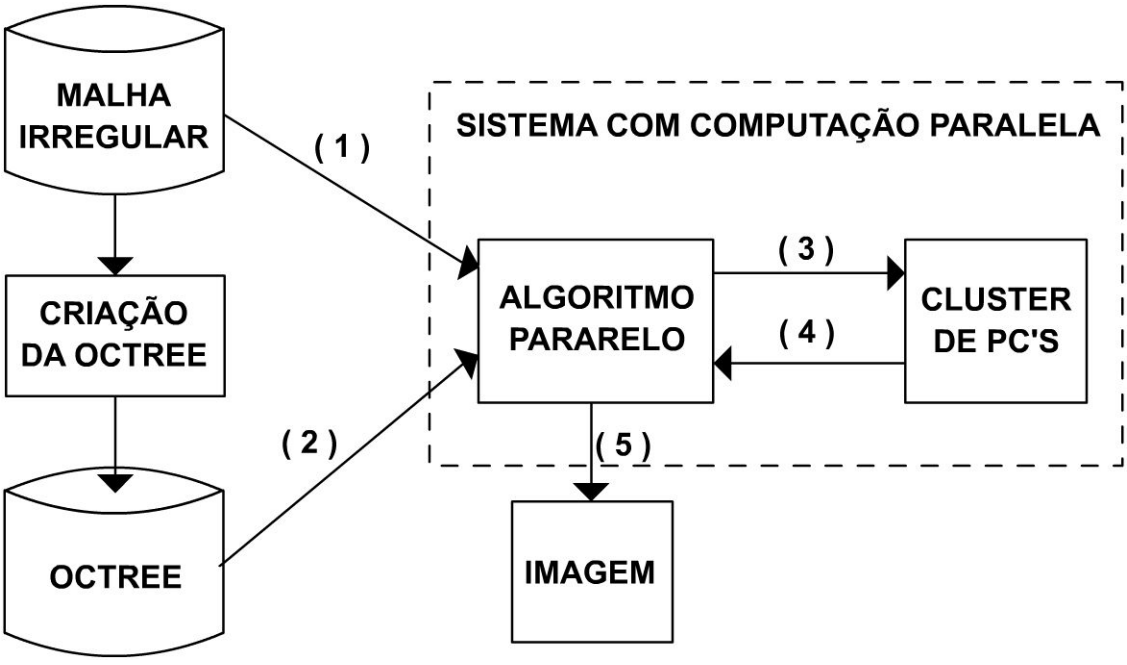

Figura 3.3: Diagrama de renderização paralela: (1) Os nós do Cluster carregam os dados da malha irregular do disco de acordo com a demanda; (2) O sistema consulta a *octree* para poder selecionar os dados corretos na malha irregular; (3) O sistema distribui as tarefas para

os nós do cluster; (4) Após o processamento, os nós do cluster devolvem os *tiles* gerados; (5) O sistema reúne os *tiles* gerados e criam a imagem final.

3.2.1. Geração da *Octree*

Com os dados de entrada no formato de uma malha irregular de voxels, a malha é logicamente subdividida segundo uma estrutura de *octree* para permitir a execução *out-of-core*. A divisão em *octree* é feita recursivamente, conforme mostra a Figura 3.4. O conjunto de dados tridimensionais é subdividido em 8 quadrantes. Cada um dos quadrantes também é dividido sucessivamente em 8 quadrantes até que a quantidade de dados de um quadrante seja menor que um determinado valor limite.

As células e vértices do conjunto de dados são distribuídos nas folhas da *octree*. Com uma busca simples na *octree* é possível determinar que folhas se projetam dentro da região da tela associada a uma determinada região da imagem, determinando assim os vértices e células dentro desta região.

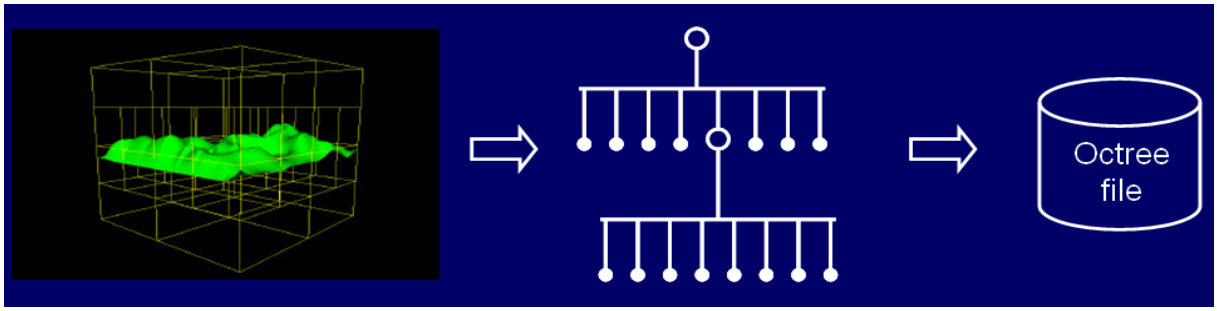

Figura 3.4: Geração da *octree*

3.2.2. Balanceamento de Carga

Um dos principais problemas enfrentados por algoritmos paralelos de visualização volumétrica é o balanceamento da carga de trabalho, dado que não se pode determinar de antemão o custo do *rendering* de uma região do volume. Determinadas regiões são muito mais custosas que outras. O balanceamento de carga tem influência direta no desempenho da renderização paralela. Em um sistema bem balanceado, os processadores terminam a renderização em tempos bem próximos e o tempo de execução se aproxima do tempo médio de renderização. Em um sistema desbalanceado, um processador pode terminar sua

renderização em um tempo próximo de zero, se a ele forem atribuídas apenas áreas vazias da tela, enquanto que outro pode terminar sua renderização, por exemplo, com o dobro do tempo médio de renderização.

O sistema DPZSweep distribui um grupo de *tiles* para cada processador renderizar e pode empregar 3 estratégias diferentes de balanceamento dinâmico de carga:

- *Longest Queue* (Fila mais longa)**:** Neste algoritmo, cada processador, ao terminar de processar a sua fila de tarefas, envia uma mensagem de solicitação de tarefas ao processador com a mais longa fila de tarefas, que lhe envia metade de suas tarefas. O processador solicitante recebe as tarefas, adiciona-as à sua fila, e continua o processamento das tarefas recebidas. Este processo, ilustrado na Figura 3.5, se repete até que todas as tarefas de todos os processadores tenham sido processadas. Este algoritmo utiliza um token circulando pela rede que contém a informação de carga de cada processador. Cada vez que o token chega num processador p, este atualiza com o número recente de *tiles* a serem processados. A informação do token pode não ser a mais atualizada, mas é próxima dela, visto que um sistema distribuído não possui informação global sobre a carga de cada elemento de processamento.

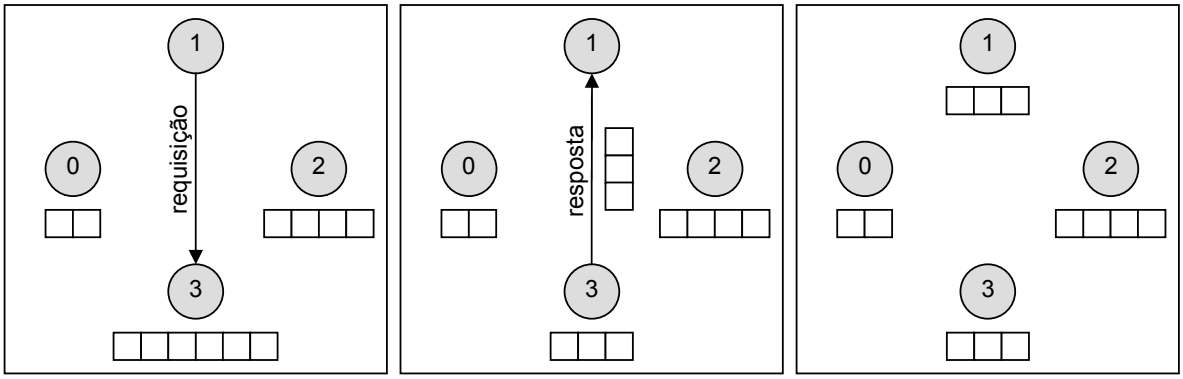

Figura 3.5: Algoritmo da Fila mais Longa: (a) O processador 1 envia uma requisição ao processador 3, que possui a fila de tarefas mais longa. (b) O processador 3 envia como resposta metade das tarefas da sua fila. (c) A carga de trabalho é homogeneamente redistribuída entre os processadores 1 e 3.

- *Circular Distribution* (Distribuição Circular): Neste algoritmo, um *token* contendo o número da próxima tarefa a ser executada circula entre os processadores, através de um mecanismo de *token-ring*. Cada processador que esteja com a fila de tarefas vazia, ao receber o *token*, deve adicionar a tarefa referenciada no *token* à sua fila de tarefas, incrementar o valor do *token* e repassá-lo ao próximo processador. Este processo é ilustrado na Figura 3.6.

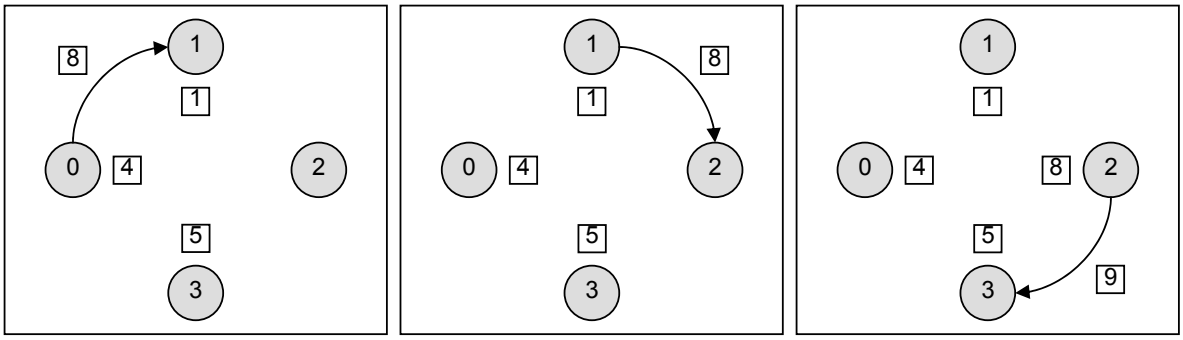

Figura 3.6: Algoritmo da Distribuição Circular: (a) O processador 0 repassa ao processador 1 o *token* contendo o número da próxima tarefa a ser executada. (b) O processador 1 repassa o *token* ao processador 2, cuja fila de tarefas encontra-se vazia. (c) O processador 2 recebe o *token*, acrescenta a tarefa indicada à sua fila, incrementa o número da próxima tarefa a ser executada, e repassa o *token* ao processador 3.

- *Nearest Neighbor:* Neste algoritmo há trocas de *tiles* entre processadores vizinhos. Quando um processador *p* acaba de processar sua fila de tarefas, ele procura por trabalho no processador *p* + 1, de acordo com a Figura 3.7.

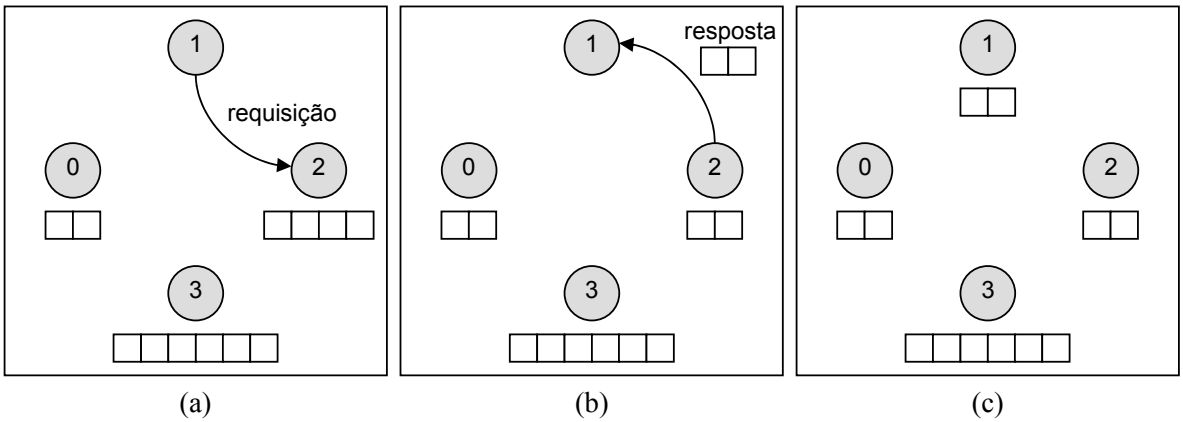

Figura 3.7: Algoritmo do Vizinho mais Próximo: os círculos representam processadores e os quadrados representam tarefas na fila de cada processador. (a) O processador 1 envia uma requisição ao vizinho mais próximo 2. (b) O processador 2 envia como resposta metade das

tarefas da sua fila. (c) A carga de trabalho é homogeneamente redistribuída entre os processadores 1 e 2.

Neste trabalho, estamos particularmente interessados na estratégia *Nearest Neighbor*. Esta estratégia é bastante promissora na implementação do algoritmo paralelo em um cluster de SMPs. Um cluster de SMPs é um cluster em que cada nó de processamento é um elemento SMP (Symmetric MultiProcessor), ou seja, possui dois ou mais processadores compartilhando uma mesma memória. A estratégia *Nearest Neighbor* (Vizinho mais próximo) é especialmente interessante para um cluster de SMPs, porque processadores vizinhos estão no mesmo nó SMP e, portanto, compartilham a mesma memória local. Neste caso, trocas de *tiles* entre vizinhos têm custo desprezível, o que reduz o tempo de processamento. Isto fica demonstrado no Capítulo 5 deste trabalho.

CAPÍTULO 4
SISTEMA OCEANVIEW 3D

Utilizando as soluções de alto desempenho apresentadas no capítulo anterior, desenvolvemos o sistema OceanView 3D para a visualização eficiente de dados oceânicos. O sistema OceanView 3D foi concebido para ser uma ferramenta de fácil uso e que reúne a possibilidade de explorar os recursos de renderização volumétrica em hardware gráfico (GPU) e em cluster de PC's. Possui também a capacidade de importar dados oceânicos de diversos formatos bem como a geração de seqüências de imagens que garantem um maior dinamismo na visualização e análise das imagens.

4.1. Arquitetura do Sistema

A arquitetura básica do nosso sistema é mostrada na Figura 4.1. OceanView 3D é composto por 3 módulos distintos:

- Pré-processamento
- Renderização
- Interface Gráfica.

O módulo de pré-processamento é responsável pela transformação dos dados oceânicos para estruturas renderizáveis. Ele gera uma malha de tetraedros e a estrutura hierárquica de *octree* para permitir a execução *out-of-core*. O módulo de renderização gera as imagens do oceano. Este módulo permite explorar, com baixo custo, as duas principais frentes da visualização de alto desempenho, tanto pelo uso da placa gráfica, como da renderização paralela em um cluster de PCs. O módulo de interface gráfica permite um manuseio simples dos dados a serem visualizados e de suas imagens. Este módulo permite ao pesquisador a interpretação e a análise visual de características e detalhes de dados oceânicos, através das imagens renderizadas sob diferentes pontos de vista de uma determinada amostragem. Nossa idéia principal é que o foco da ferramenta é para uso por pesquisadores leigos na área de Visualização Científica e de Computação de Alto Desempenho, que precisam de imagens dos dados coletados em tempo-real.

A seguir descrevemos em detalhes cada um dos 3 módulos do sistema OceanView 3D, dando ênfase para as soluções fornecidas para os problemas de transformação dos dados, geração de imagens em tempo-real e interação homem-computador.

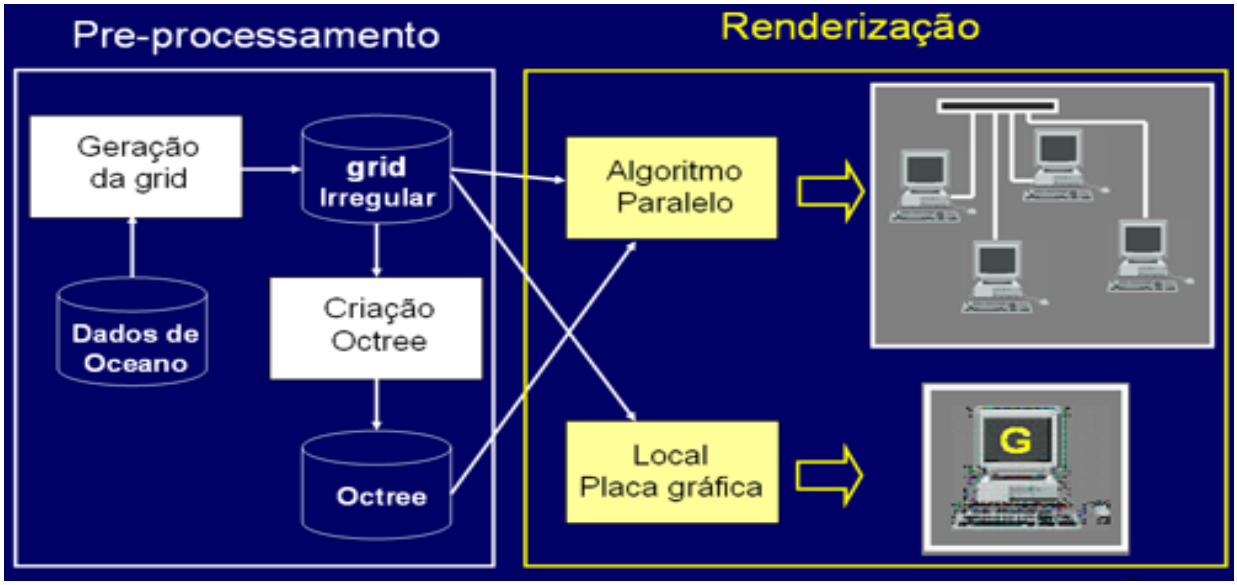

Figura 4.1: Diagrama básico da arquitetura do sistema OceanView 3D

4.1.1. Pré-processamento

O Módulo de pré-processamento foi desenvolvido na linguagem C++ e é responsável pela conversão e formatação dos dados oceânicos de forma que possam ser utilizados pelo módulo de renderização. Dividimos este módulo internamente em 2 sub-módulos:

- Conversão de dados;
- Geração da *Octree*.

Vale ressaltar que o módulo de pré-processamento é executado uma única vez para uma determinada massa de dados. Um mesmo dado pode ser visualizado em diferentes pontos de vista, ou ângulos de visão. Para cada ângulo de visão diferente, uma nova imagem deve ser gerada. Entretanto, as estruturas da malha irregular e da *octree* não se modificam. Portanto, estamos assumindo que o custo de se gerar a malha e a *octree* vai se diluir na medida que inúmeros pedidos de visualização em diferentes pontos de vista são requisitados.

4.1.1.1. Conversão de Dados

O módulo de conversão de dados foi desenvolvido para permitir a utilização de dados brutos de oceano como fonte de informações para geração de imagens. Existem diversos formatos de bases de dados oceânicos (NRL, NGTC, NOA, etc.),mas a princípio, este módulo está habilitado a converter apenas dados no formato do Navy Research Laboratory (NRL), pois este é o formato da massa de dados que foi utilizada para validar o nosso sistema. Os dados da NRL possuem uma formatação específica, uma estrutura que armazena dados pontuais com informações de latitude, longitude, profundidade e um valor escalar que representa a grandeza a ser estudada, como por exemplo: salinidade, temperatura, concentração de oxigênio, etc. Estes dados estão armazenados em arquivos binários de forma que dados de mesma profundidade são agrupados em um único arquivo, ou seja, cada arquivo representa o conjunto de dados de um nível de profundidade.

Os dados oceânicos não estão representados sob a forma de uma malha poliédrica, mas apenas de um conjunto de pontos com coordenadas (latitude e longitude) e um valor escalar (da grandeza medida) associado. O módulo de conversão de dados transforma de forma sistemática cada conjunto de 8 pontos em um hexaedro e posteriormente o divide em 5 tetraedros para que todo o conjunto de dados passe a ter a forma de uma malha irregular de tetraedros, conforme mostra a Figura 4.2. Desta forma obtemos uma malha irregular de tetraedros que é o tipo de malha que o nosso sistema é capaz de processar.

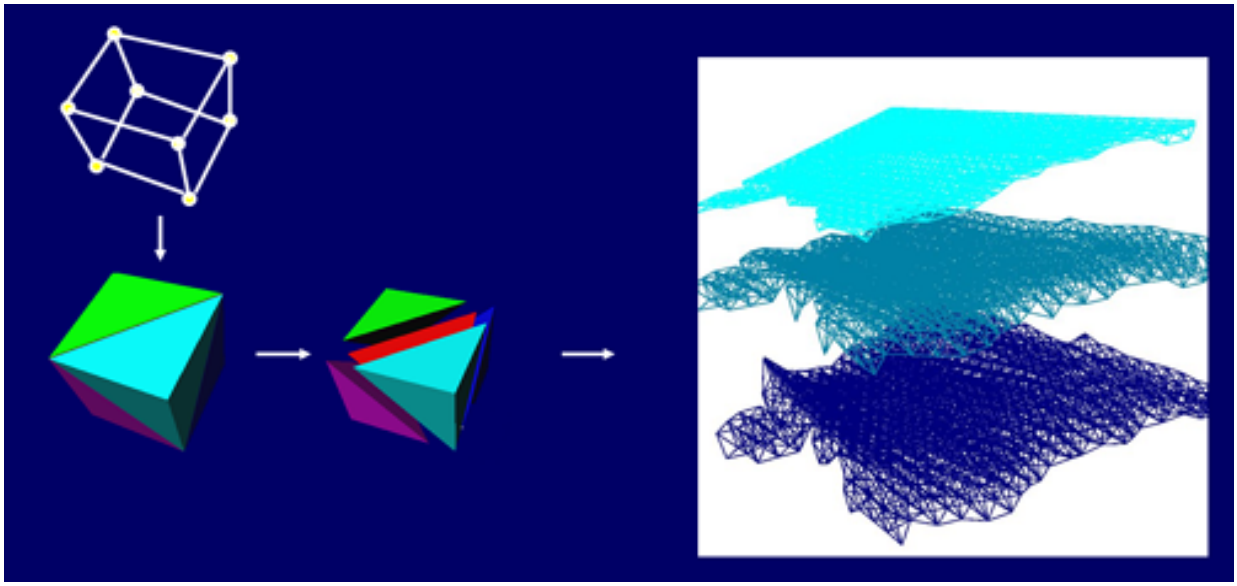

Figura 4.2: Geração da malha irregular: cada hexaedro é dividido em 5 tetraedros.

Para exemplificar o processo de conversão, suponhamos um conjunto de dados NRL cujas informações sobre os pontos amostrados são descritas na Tabela 4.1:

PONTO	COORDENADAS			VALOR ESCALAR
	X	Y	Z	
1	1.0	3.0	2.0	2.0
2	1.0	3.0	1.0	5.0
3	2.0	3.0	1.0	9.0
4	2.0	3.0	2.0	4.0
5	1.0	1.0	2.0	4.0
6	1.0	1.0	1.0	3.0
7	2.0	1.0	1.0	8.0
8	2.0	1.0	2.0	1.0

Tabela 4.1: Exemplo básico de pontos amostrados de oceano.

Na Figura 4.3 representamos como se processa a conversão dos dados da Tabela 4.1 tetraedros irregulares.

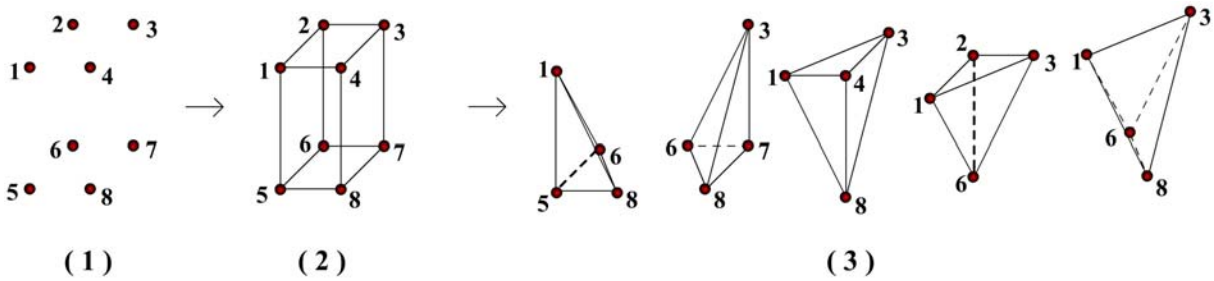

Figura 4.3: Exemplo de conversão de dados oceânicos NRL

(1) Os dados são representados pelas suas coordenadas espaciais; (2) Considera-se cada 8 pontos formando um hexaedro; (3) Divide-se o hexaedro em 5 tetraedros irregulares.

O conjunto dos tetraedros resultantes constituem a malha irregular, que no exemplo da Figura 4.3 é constituída por 5 tetraedros. Esta malha irregular fica armazenada em um arquivo ASCII cuja extensão é ".off". O formato OFF (Object File Format) [Rost86] foi desenvolvido pela WSE (Workstation Systems Engineering) para o intercâmbio e armazenamento de objetos tridimensionais. Este formato possui versões binárias, que possuem maior velocidade de leitura e escrita, e versões em ASCII, que possuem a finalidade de intercâmbio.

As linhas de um arquivo ".off" em ASCII são usadas para representar a geometria de um corpo, especificando os polígonos que o constituem. No caso do nosso sistema, os polígonos que constituem o corpo são tetraedros irregulares e são descritos no arquivo ".off" de acordo com o exemplo da Figura 4.4.

```
8 >(1)
5 >(2)
1.0 3.0 2.0 2.0
1.0 3.0 1.0 5.0
2.0 3.0 1.0 9.0
2.0 3.0 2.0 4.0
1.0 1.0 2.0 4.0   } (3)
1.0 1.0 1.0 3.0
2.0 1.0 1.0 8.0
2.0 1.0 2.0 1.0
1 5 6 8
3 6 7 8
1 3 4 8   } (4)
1 2 3 6
1 3 6 8
```

Figura 4.4: Conteúdo de um exemplo de arquivo ".off" resultante da conversão de dados NRL
(1) Esta linha descreve quantos pontos (vértices) possui o corpo; (2) Esta linha descreve quantos tetraedros possui o corpo; (3) Nestas linhas são descritas as coordenadas e valor escalar de cada ponto do corpo. Neste exemplo, os as informações são provenientes da Tabela 4.1; (4) Nestas linhas são enumerados os pontos que formam cada tetraedro que compõem o corpo. Neste exemplo, as informações são provenientes da Figura 4.3.

Na Figura 4.5 exibimos a imagem resultante do processamento do arquivo ".off" exemplificado. Podemos observar as variações de cores em função dos valores escalares e o formato de hexaedro do corpo.

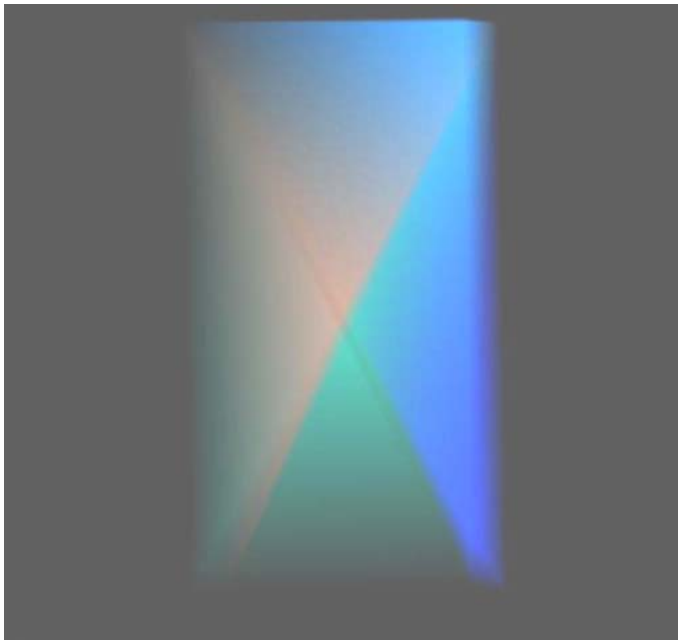

Figura 4.5: Renderização de volume do arquivo ".off" de exemplo.

Dada a natureza modular do sistema OceanView 3D, de fato, podemos utilizar qualquer tipo de dados oceânicos como entrada para a visualização, bastando para isso, construir um módulo de conversão específico para cada tipo de formato que gera a malha de tetraedros.

4.1.1.2 Geração da *Octree*

O arquivo que contém a malha irregular pode ter tamanhos diversos em função do número de pontos e tetraedros que retratam o trecho de oceano. Como a quantidade de dados provenientes do oceano pode gerar malhas irregulares que podem não caber na memória do computador, foi utilizado um módulo proveniente do DPZSweep de criação de *octree* que servirá de índice de busca para os dados da malha irregular armazenados em disco. A *octree* criada fica armazenada em disco num arquivo binário cuja extensão é ".oct".

Para melhor entendimento, vamos considerar um corpo em formato esfera onde dividimos o seu espaço envolvente em oito quadrantes (A, B, C, D, E, F, G, H). Se selecionarmos o quadrante "D" e o dividirmos em outros 8 quadrantes (a, b , c, d, e, f, g, h), teremos a possibilidade de construir uma estrutura hierárquica em árvore que represente estas divisões de acordo com a Figura 4.6. Esta estrutura denomina-se *octree* e desta forma podemos localizar dados no interior do corpo, utilizando como referência os quadrantes em que o corpo foi subdividido. O nó do topo da *octree* representa todo o espaço ocupado pelo corpo. Os nós do nível abaixo do topo representam a divisão do corpo em 8 quadrantes e cada nível inferior representa as divisões em quadrantes dos nós do nível imediatamente superior. O número de níveis vai variar de acordo com o tamanho da massa de dados de forma a garantir uma busca rápida e eficaz.

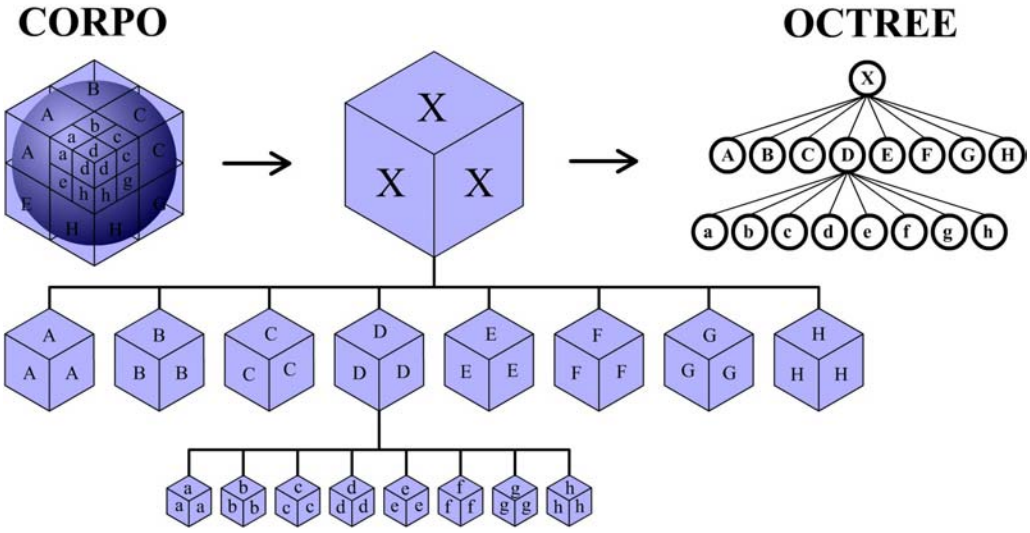

Figura 4.6: Divisão de um corpo em quadrantes para criação da *octree.*

Após a criação da *octree*, os dados da malha irregular que estão armazenados no arquivo ".off" são reestruturados de acordo com o estabelecido na *octree* e é gerado um novo arquivo binário que contém os dados da malha irregular. Este arquivo possui a extensão ".dat" e os informações dos tetraedros pertencentes a cada nó da *octree* são agrupados de acordo com a hierarquia estabelecida. É neste arquivo que serão efetuadas as buscas de dados para o processamento paralelo.

A geração da *octree* para execução *out-of-core* é necessária apenas para a renderização paralela, já que a renderização na placa gráfica requer que os tetraedros estejam na memória. Na renderização paralela, as informações contidas no arquivo ".oct" são carregadas em memória, de modo que a estrutura da *octree* fica carregada na memória durante toda a execução do algoritmo. Toda vez que um processador vai renderizar um tile T, ele busca na *octree* quais as folhas da *octree* possuem dados dentro dos limites de T. Apenas os tetraedros que compõem estas folhas são trazidos para a memória na renderização de T.

4.1.2. Módulo de Renderização

O módulo de renderização é responsável pela geração de imagens a partir de dados oceânicos no formato de malhas irregulares e é neste módulo que ocorrem as operações que mais exigem recursos computacionais para processamento.

Para que o usuário do sistema tenha a sua disposição 2 excelentes métodos de renderização, este módulo é constituído por 2 sub-módulos. São eles:

- Renderização paralela;
- Renderização local em hardware gráfico;

Estes sub-módulos são resultados dos trabalhos anteriores citados em [Coelho04a] e [Maximo06] e foram todos escritos em C++.

Uma das principais vantagens do sistema OceanView 3D está na flexibilidade de uso da renderização de alto desempenho. Normalmente, as ferramentas de visualização volumétrica optam por atacar o problema de desempenho, na geração de imagens em tempo-real, de uma única forma, seja utilizando processamento paralelo, seja utilizando os benefícios da placa gráfica. Estamos defendendo neste trabalho que nenhuma das duas formas de se resolver o problema de desempenho é ideal para todas as situações.

A solução com o uso da placa gráfica provê tempos interativos com taxas de várias imagens geradas por segundo. É a solução mais eficiente, porque não envolve

comunicação remota. Toda a renderização é executada localmente. Esta solução, entretanto, tem dois grandes limitadores: a existência de uma placa gráfica compatível com o código e o tamanho da memória da placa gráfica. A programação da placa gráfica tem se tornado cada vez mais genérica e flexível, entretanto, um código escrito para uma GPU de determinado fabricante, não executa na GPU de outro fabricante, visto que as instruções da GPU são limitadas. O código apresentado no Capítulo 3, embora bastante eficiente e com resultados promissores, requer que os tetraedros da massa de dados caibam na memória da placa gráfica, visto que eles serão inicialmente ordenados pela coordenada z do centróide dos tetraedros. Portanto, ele não pode renderizar dados ocânicos que não caibam na memória disponível na placa gráfica utilizada. Dados maiores que 256Mb são considerados grandes e não podem ser processados pelas placas gráficas atuais.

A solução paralela, aproveita o uso de arquiteturas de cluster de PCs e pode até ser estendido para aproveitar a computação oportunista de uma *grid*. O uso do cluster, ou da *grid*, entretanto, implica no overhead de distribuição da massa de dados pelos nós de processamento e de se recolher os resultados finais. Para massas de dados muito grandes, entretanto, é a única solução viável para a visualização em tempo-real, dado que implementa a renderização *out-of-core*.

Portanto, um dos grandes trunfos da ferramenta que estamos propondo é permitir que o desempenho seja obtido, independente da arquitetura da máquina que ela executa ou do tamanho dos dados. Quando a máquina possui placa gráfica NVidia e os dados não ocupam mais do que 256M, então a renderização executa localmente com imagens geradas em tempo-real. Quando a máquina utilizada não possui estas características, ou o dado é maior que a capacidade da memória da GPU, então a ferramenta se conecta a um cluster (em que o usuário possua conta) e executa a renderização de forma paralela.

4.1.2.1 Renderização Paralela

O módulo de renderização paralela gera imagens a partir dos dados oceânicos utilizando processamento paralelo em um cluster de PC's. Ele é responsável pela divisão e distribuição de tarefas e o balanceamento de carga entre os nós do cluster.

Utilizamos o algoritmo DPZSweep para executar a renderização paralela em cluster de PC's por possuir um bom desempenho e processamento *out-of-core*. No algoritmo paralelo, cada nó do cluster lê o arquivo da *octree* (".oct") para a memória e processa os *tiles* que lhe foram designados.

Para cada tile a ser computado, o processador busca no arquivo ".dat" os tetraedros correspondentes às folhas da *octree* que pertencem ao tile em questão. Estes tetraedros são utilizados no algoritmo sequencial de renderização Zsweep para gerar a porção da imagem relativa ao tile em questão. As porções da imagem computadas de forma distribuída são coletadas por um único processador que as reúne para montar a imagem final, que será exibida pelo módulo de interface gráfica.

A utlização deste submódulo é recomendada para dados oceânicos de grandes proporções (maiores que 256Mb).

4.1.2.2 Renderização Local em Hardware Gráfico

Este sub-módulo gera imagens a partir dos dados oceânicos armazenados no arquivo ".off" utilizando o poder de processamento gráfico das placas de vídeo 3D da NVidia. Utilizando instruções OpenGL para executar a projeção das faces dos tetraedros, este submódulo obtém excelentes desempenhos na geração das imagens de oceano. Sua limitação está na capacidade da memória da placa de vídeo utilizada que está atualmente em 256Mb.

A utilização deste sub-módulo é recomendada para dados oceânicos de pequenas e médias proporções.

4.1.3 Interface Gráfica

O especialista que analisará as imagens de oceano não precisa necessariamente ter conhecimentos profundos de Computação Gráfica ou Paralela para utilizar um sistema visualização, pois é desejável que um sistema computacional seja dinâmico, intuitivo e de fácil utilização.

Originalmente, os módulos de renderização utilizados no nosso sistema possuem como interface com o usuário apenas a linha de comando do sistema operacional. Isto torna pouco agradável a utilização destes sub-módulos. Buscamos, então, criar uma interface gráfica que permitisse uma integração confortável entre o usuário e os módulos de pré-processamento e de renderização. Esta interface além de amigável, possui tratamento de possíveis erros cometidos pelo usuário (verificação e solicitação de confirmação de alteração em arquivos já existentes). Na Figura 4.7, é exibida a janela inicial da interface gráfica utilizada.

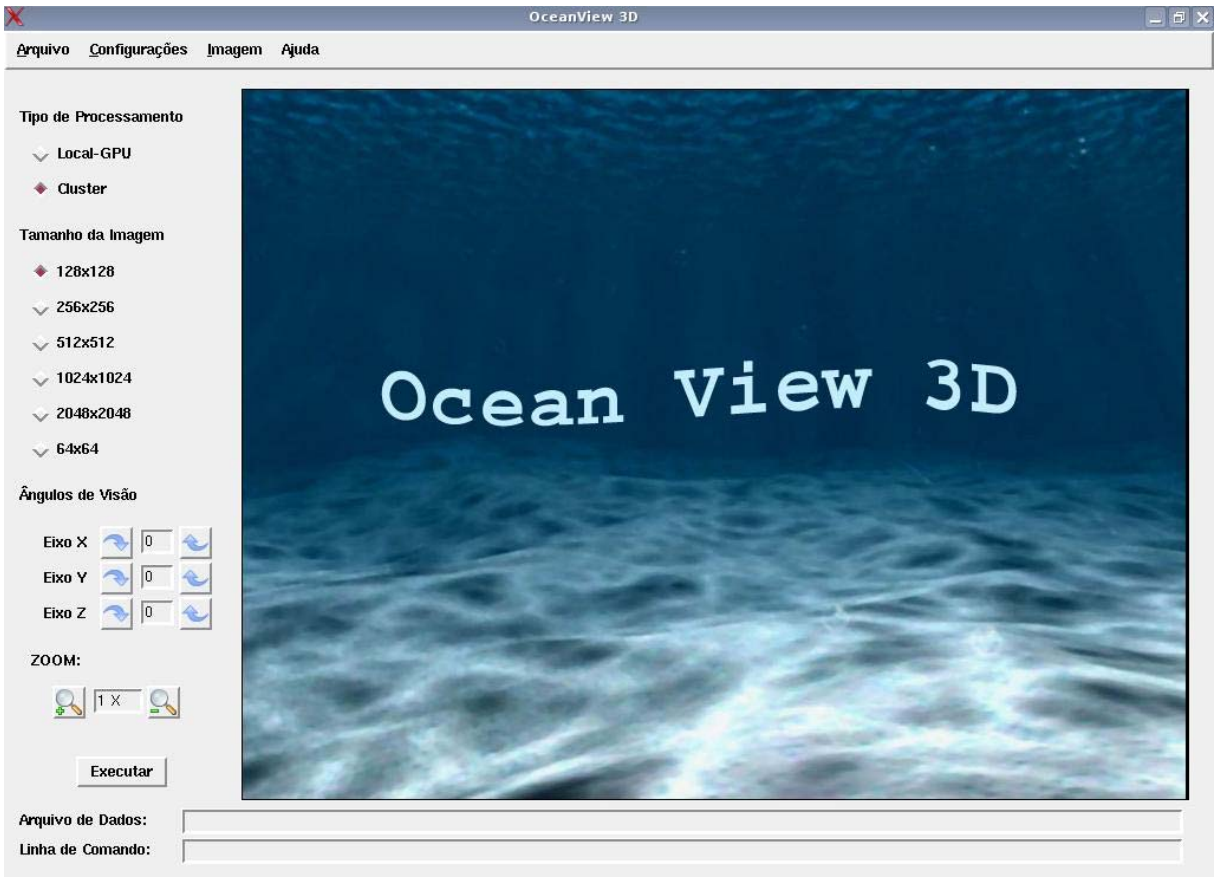

Figura 4.7: Janela Inicial da Interface.

4.1.3.1 Tecnologia Utilizada

Como nosso sistema tem como objetivo utilizar recursos computacionais de baixo custo, todo o sistema foi concebido para utilizar software livre, incluindo as ferramentas de desenvolvimento e o sistema operacional Linux.

Para desenvolver o módulo de interface gráfica no Linux, escolhemos a linguagem TCL (Tool Command Language) entre as diversas alternativas de linguagens de programação por ser uma linguagem simples, genérica e extensível e que possui uma biblioteca gráfica (TK – Tool Kit) que oferece um conjunto de recursos gráficos reutilizáveis para Linux. Esta mesma biblioteca permite trabalhar com imagens PPM (*Portable Pixmap*) que é o formato da imagem gerada pelo módulo de renderização paralela.

A linguagem TCL foi desenvolvida por John Ousterhout no final da década de 1980 [Franca05] e é amplamente utilizada na atualidade para desenvolver aplicações gráficas para Linux. Por ser uma linguagem interpretada, não há a necessidade de compiladores, todo o código fonte é lido e processado por um interpretador. Os programas interpretados são mais lentos que os compilados, mas isto não é crítico para a construção de nossa interface gráfica, pois os recursos para programação gráfica que a linguagem oferece são adequados.

Além de integrar os módulos de pré-processamento e de renderização, foram inseridas novas funcionalidades que permitem explorar ainda mais a visualização científica de oceanos. Nossa idéia é prover a flexibilidade ao usuário com relação à exploração de mecanismos de alto desempenho e também de permitir o armazenamento de imagens pré-geradas para uma futura visualização na forma de animação. Este armazenamento funciona como uma *cache* de imagens geradas. Utilizando esta *cache*, evitamos ter que renderizar um mesmo dado mais de uma vez, dado que já foi mostrado que a renderização é computacionalmente muito custosa.

4.2 Organização de Arquivos

O Sistema OcenView 3D é composto por diretórios e arquivos em disco e está organizado em numa estrutura de diretórios, conforme mostra a Figura 4.8 .

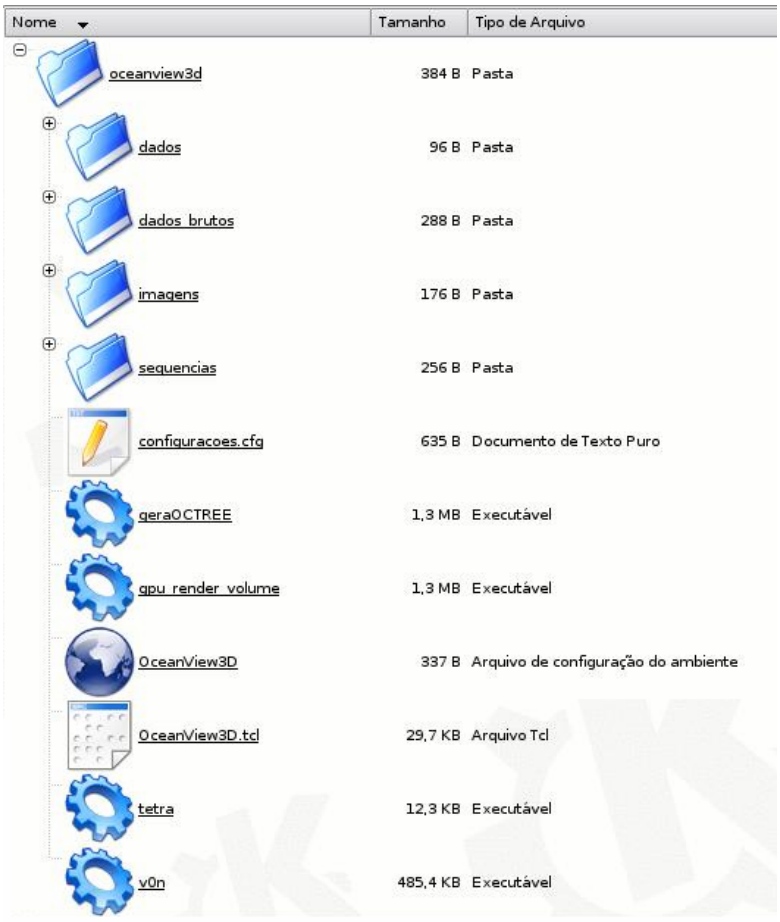

Figura 4.8: Estrutura de diretórios do sistema OceanView 3D

- Diretório "oceanview3d": Este é o diretório principal do sistema, nele ficam armazenados os demais diretórios, os arquivos executáveis dos módulos e o arquivo de configuração do sistema;

- Diretório “dados_brutos”: Neste diretório estão o diretórios onde estão armazenados os dados oceânicos que serão convertidos para o formato padrão do sistema;

- Diretório “dados”: Neste diretório estão o diretórios onde são armazenados os dados já convertidos e prontos para serem utilizados pelo sistema;

- Diretório “imagens”: Neste diretório estão o diretórios onde são armazenadas as imagens geradas pelo sistema.

- Diretório “sequencias": Neste diretório estão o diretórios onde são armazenadas as seqüências de imagens geradas pelo sistema. A geração de seqüências de imagens será explicada no ítem 4.3 deste capítulo.

Além da estrutura de diretório apresentada, o sistema OceanView 3D possui alguns arquivos importante para o seu funcionamento, são eles:

- “configuracoes.cfg”: Neste arquivo são gravadas as configurações do sistema. Por ser um arquivo no formato ASCII, é possível a sua edição em qualquer editor de textos. Segue abaixo o conteúdo típico do arquivo:

```
#Este arquivo armazena as configurações utilizadas pelo OceanView 3D

#parâmetros do sistema paralelo em cluster de pc's

#dimensões dos tiles (16 ou 32 para 16x16 ou 32x32)
tam_tiles=16

#número de nós do cluster
n_nos=2

#brilho da imagem
brilho=3
```

- “oceanview.tcl”: Este é o arquivo que contém o script da IHM do nosso sistema. Para utilizarmos o sistema, devemos executar este arquivo;

- “v0n” : Este é o arquivo executável responsável pelo módulo de renderização paralela;

- “gpu_render_volume”: Este é o arquivo executável responsável pelo módulo de renderização local em placa gráfica;

- “conversorNRL”: Este é o arquivo executável responsável pelo módulo de conversão de dados oceânicos do formato NRL.

- “geraOCTREE”: Este é o arquivo executável responsável pelo módulo de criação da octree.

4.3 Funcionalidades do Sistema OceanView 3D

O sistema OceanView 3D agrega funcionalidades que auxiliarão o especialista na obtenção de maior produtividade. A seguir apresentamos cada uma dessas funcionalidades.

Seleção de arquivos para conversão

Para converter dados oceânicos para o formato de malhas irregulares, basta selecionar os arquivos a serem convertidos diretamente pela interface gráfica e informar o nome do arquivo “.off” que será gerado.

Geração de imagens

O sistema gera imagens de acordo como o arquivo de dados e método de renderização escolhidos. A imagem gerada é automaticamente exibida pelo sistema que permite a movimentação e o zoom da mesma. Além disso, o sistema permite que o usuário determine a precisão da imagem gerada e o ângulo de visão.

Visualização de imagens PPM armazenas em disco

As imagens de oceanos previamente geradas e gravadas em disco poderão ser abertas diretamente no sistema. Como recurso básico de visualização, é fornecida a possibilidade de movimentação da imagem na tela e ajuste de zoom. Dessa forma, podemos montar uma animação com as várias imagens geradas com a mesma massa de dados. Na verdade, esta funcionalidade do sistema funciona como uma *cache* de imagens já renderizadas e evita que uma renderização custosa seja executada mais de uma vez.

Salvar imagens

Toda imagem gerada ou visualizada no sistema pode ser copiada. Esta imagem terá o formato PPM e poderá ser salva no local de preferência do usuário.

Edição de imagens

Caso seja necessária a edição das imagens que estão sendo visualizadas, foi incluída no sistema a possibilidade de abertura da imagem diretamente no editor Gimp que acompanha as distribuições Linux.

Geração automática de seqüências de imagens

Dependendo da quantidade de dados oceânicos, a geração de uma imagem pode levar um tempo considerável e o usuário pode estar interessado em visualizar uma seqüência de imagens que retrate uma rotação nos eixos. Para que o usuário não precise gerar uma imagem por vez, foi implementada no sistema a funcionalidade de geração automática de seqüências de imagens, onde o usuário fornece o intervalo de ângulos de rotação nos 3 eixos e o número de imagens a serem obtidas e o sistema se encarrega de obtê-las. Desta forma o usuário pode, por exemplo, efetuar a análise de uma imagem enquanto o sistema gera as outras.

Configuração do Sistema

O sistema pode ser configurado por meio da interface gráfica, sem a necessidade de edição do arquivo de configuração. Isto possibilita que o usuário alterne as configurações até obter a melhor eficiência possível.

Manual de utilização embutido

Para facilitar ainda mais a utilização do sistema pelo usuário, é possível visualizar o manual de utilização do OceanView 3D diretamente pelo menu ajuda. Este manual contém toda a informação necessária para que o especialista utilize de forma produtiva os recursos do sistema.

4.4 Utilizando o Sistema OceanView 3D

Abordaremos agora a utilização prática do sistema OceanView 3D, exibindo passo-a-passo como explorar as funcionalidades do sistema.

4.4.1. Importando dados oceânicos

Selecione a opção **Importar Dados** do menu **Arquivo**, conforme Figura 4.9. Na janela exibida, selecione os arquivos desejados de acordo com a Figura 4.10 e posteriormente informe um nome para o arquivo de destino.

Figura 4.9: Selecionando Importar no menu Arquivo

Figura 4.10: Janela para importar arquivos de dados no formato NRL

4.4.2. Abrindo dados oceânicos

Selecione a opção Abrir Arquivo de Dados no menu Arquivo, conforme Figura 4.11. Na janela exibida, selecione o arquivo que contém os dados oceânicos ".off", conforme Figura 4.12.

Figura 4.11: Selecionando Abrir Arquivo de Dados no menu Arquivo

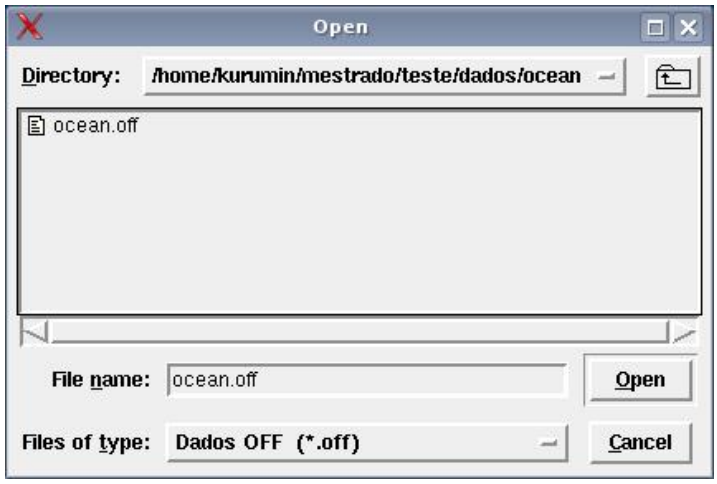

Figura 4.12: Janela para abrir arquivos de dados no formato ".off"

4.4.3. Configurando o Sistema de Renderização Paralela

Selecione a opção Cluster do menu Configurações, conforme Figura 4.13. Na janela exibida, selecione as opções desejadas e clique em salvar, vide Figura 4.14.

Figura 4.13: Selecionando Cluster no menu Configurações

Figura 4.14: Janela para configurações utilizar o Cluster

4.4.4. Gerando imagens a partir de dados oceânicos num cluster

Após abrir o arquivo de dados oceânicos e selecionar a configuração para renderização no cluster, devemos selecionar o Tipo de Processamento em Cluster, escolher o tamanho da imagem a ser gerada, os ângulos de visão tridimensional e clicar no botão Executar, conforme a Figura 4.15. A imagem resultante será exibida na tela do sistema conforme a Figura 4.16.

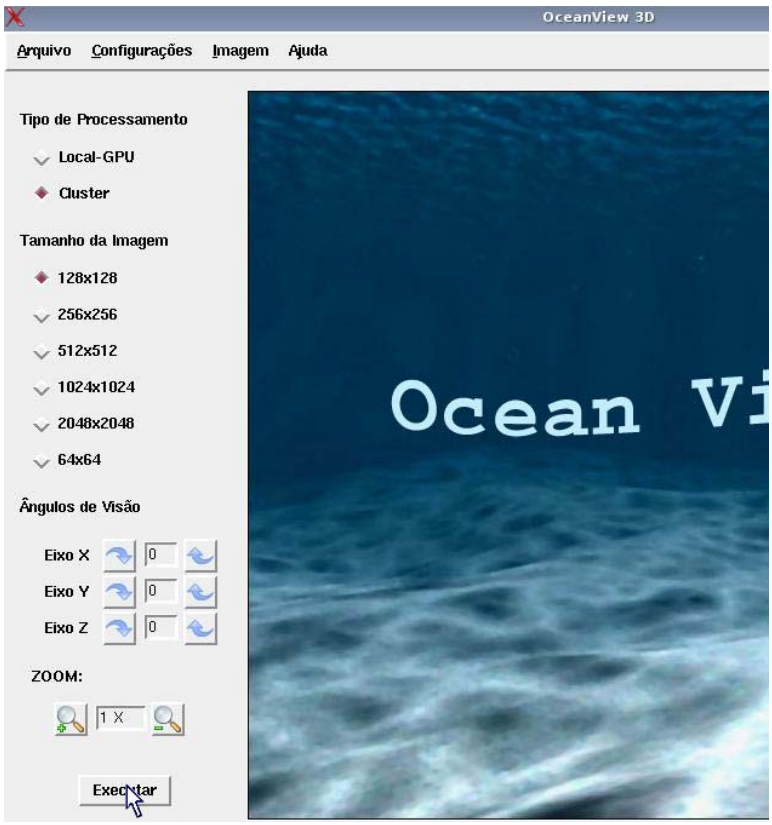

Figura 4.15: Campos para configuração da renderização

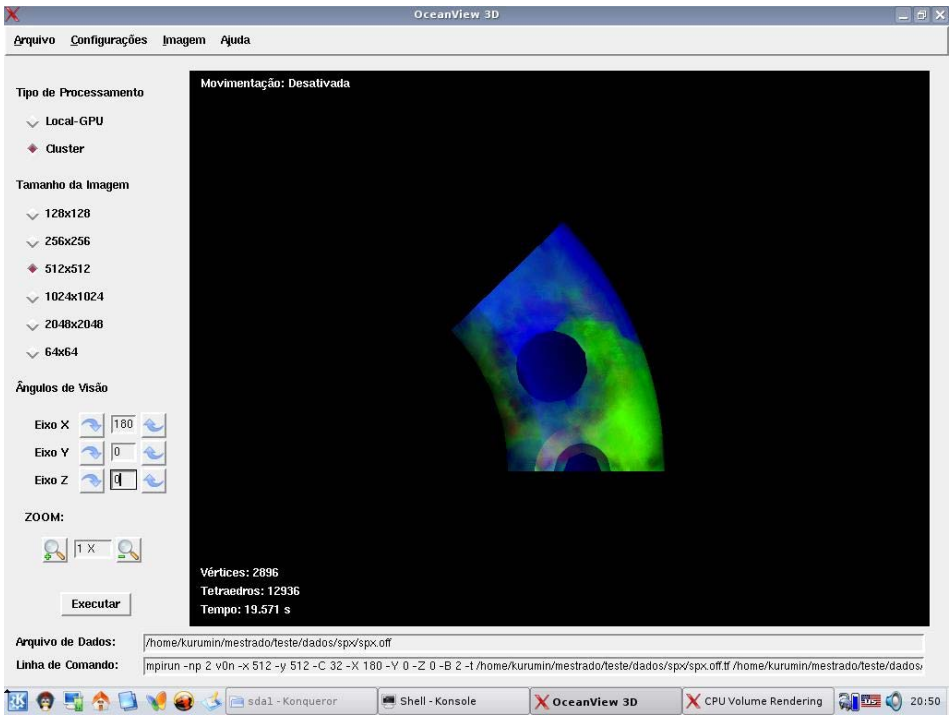

Figura 4.16: Imagem resultante de processamento em Cluster do dado SPX (dado para teste de renderização) exibida na tela do sistema.

4.4.5. Gerando Imagens Localmente com Hardware Gráfico

Após abrir o arquivo de dados oceânicos e selecionar o Tipo de Processamento em Local – GPU, basta clicar em Executar que será exibida a janela de visualização local com hardware gráfico conforme a Figura 4.17.

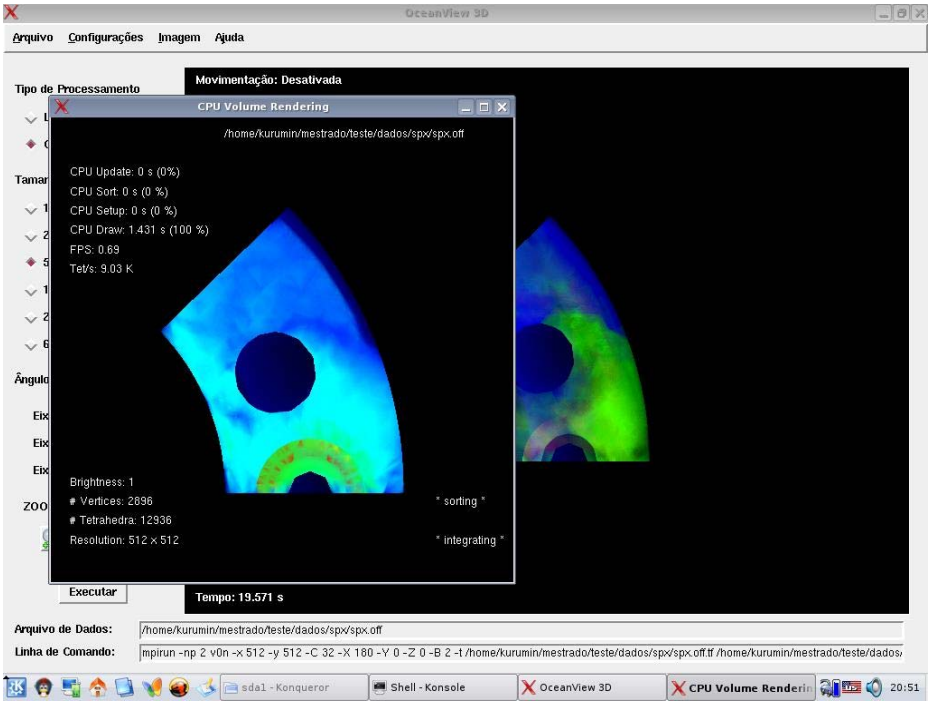

Figura 4.17: Imagem resultante de processamento local com hardware gráfico.

4.4.6. Abrindo uma Imagem PPM para Visualização

Selecione a opção Abrir Imagem do menu Imagem, conforme Figura 4.18. Na janela exibida, selecione o arquivo PPM desejado e clique em Open conforme Figura 4.19. O sistema exibirá a imagem conforme a Figura 4.20.

Figura 4.18: Selecionando Abrir Imagem no menu Imagem

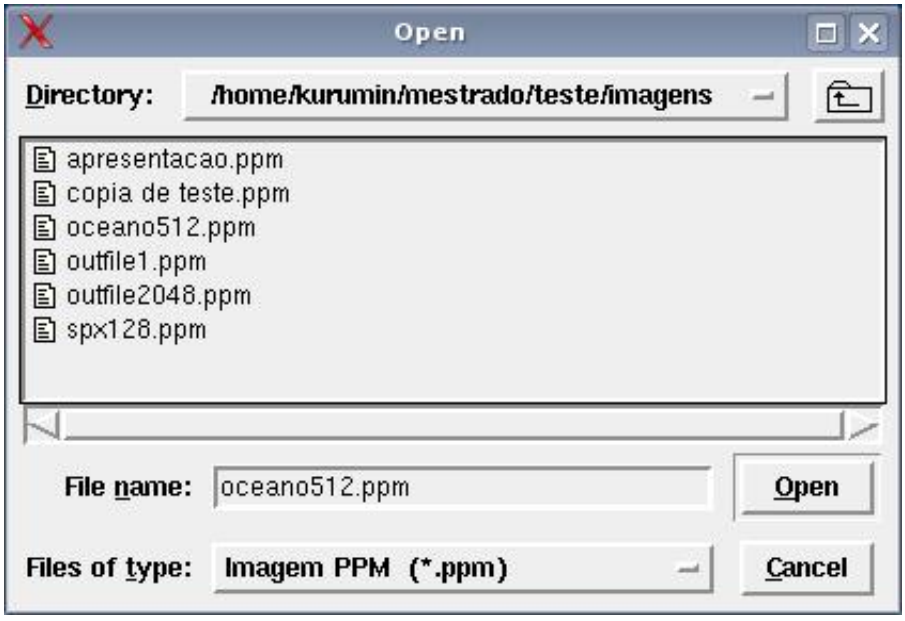

Figura 4.19: Janela para abrir arquivos de imagens no formato PPM

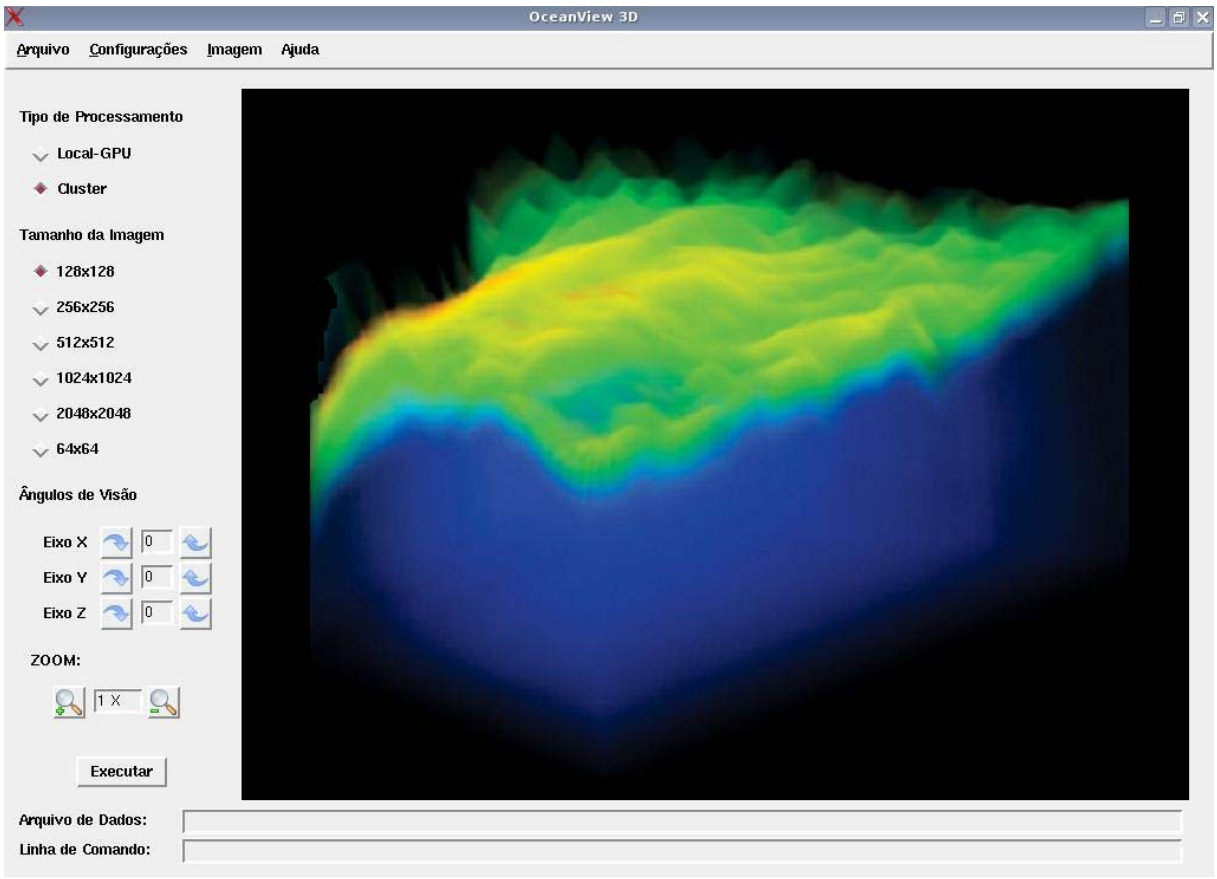

Figura 4.20: Visualizando a imagem

4.4.7. Editando uma imagem

Para editar uma imagem que está sendo visualizada, basta selecionar a opção Editar Imagem no menu Imagem conforme Figura 4.21. A imagem será automaticamente aberta no editor Gimp que acompanha a distribuição Linux, vide Figura 4.22.

Figura 4.21: Selecionando Editar Imagem no menu Imagem

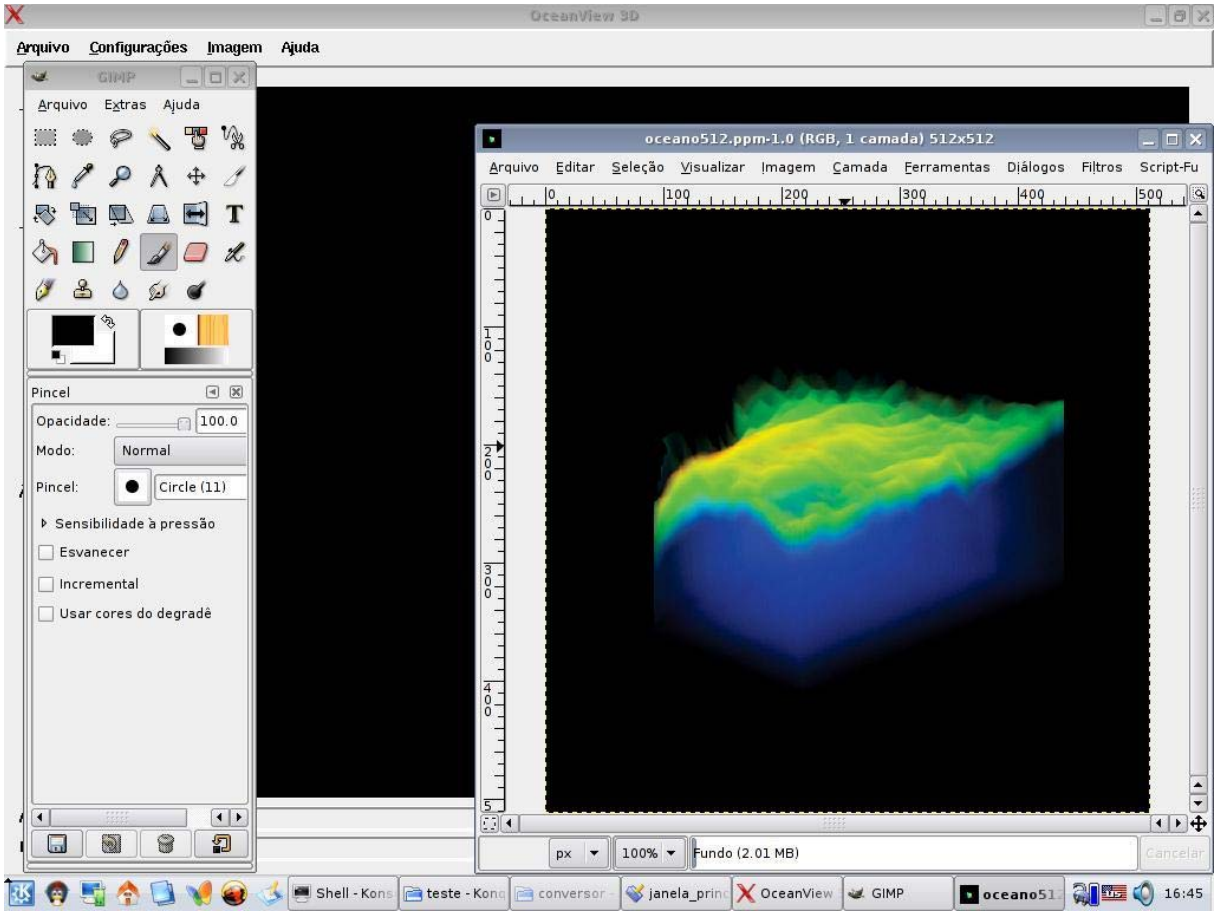

Figura 4.22: Imagem aberta para edição no Gimp

4.4.8. Abrindo Seqüências de Imagens

Para abrir uma seqüência de imagens do disco, selecione a opção Seqüências no Menu Imagem para que seja aberta a janela do Gerador e Visualizador de Seqüências, conforme Figura 4.23. Selecione a opção Abrir Seqüência de Imagens no menu Arquivo desta janela, conforme Figura 4.24. Para avançar ou voltar a seqüência de imagens, pode-se utilizar as setas de movimentação ou o movimento do mouse, conforme ilustrado na Figura 4.25.

Figura 4.23: Selecionando Seqüência de Imagens no Menu Imagem

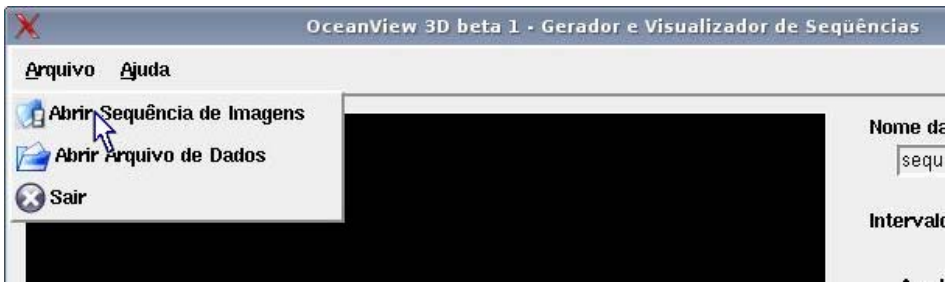

Figura 4.24: Selecionando Abrir Seqüência de Imagens no menu Arquivo da Janela do Gerador e Visualizador de Seqüências

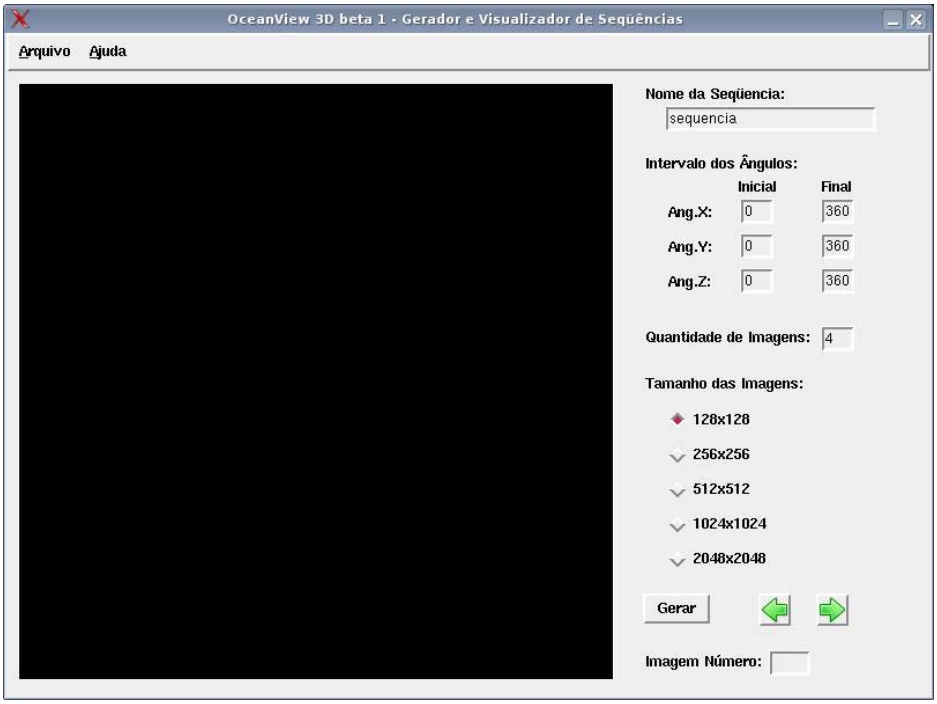

Figura 4.25: Janela do Gerador e Visualizador de Seqüências

4.4.9. Gerando Seqüências de Imagens

Com a janela do Gerador e Visualizador de Seqüências aberta, preencha o formulário com um nome de base para a seqüência, os intervalos de ângulos de rotação dos eixos X, Y e Z, a quantidade de imagens a serem geradas no intervalo, o tamanho padrão das imagens geradas e clique no botão Gerar, conforme Figura 4.26. Após o término da geração, a primeira imagem da seqüência será exibida na tela, conforme Figura 4.27. Todas as seqüências de imagens são gravadas no diretório “sequências” em um subdiretório com o nome base da seqüência.

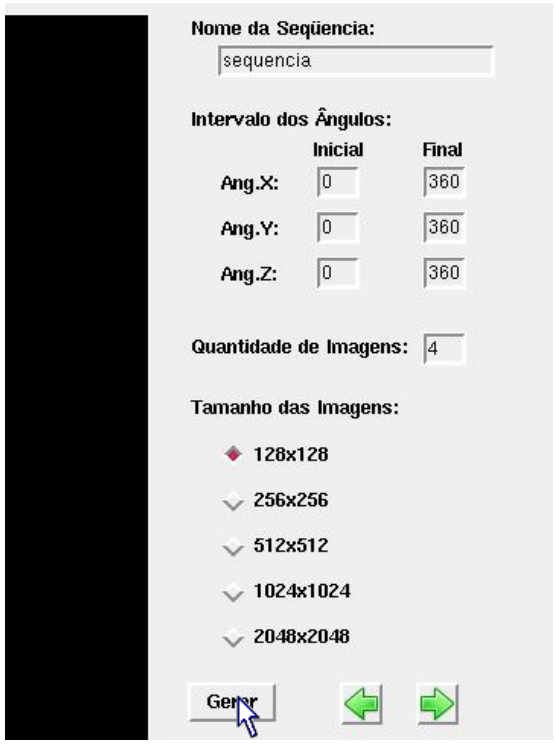

Figura 4.26: Formulário para geração de seqüências

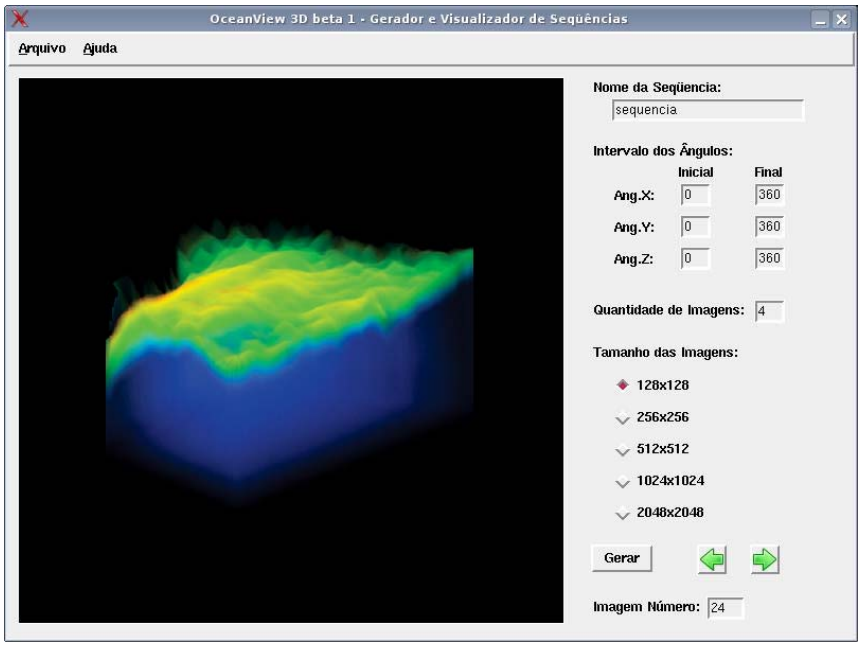

Figura 4.27: Imagem na janela do Gerador e Visualizador de Seqüências

4.4.10. Salvando Imagens no Disco

Para salvar uma imagem que está sendo visualizada, basta selecionar a opção Salvar Cópia do menu Imagem, conforme Figura 4.28 e escolher o local onde ela será salva por meio da janela mostrada na Figura 4.29.

Figura 4.28: Selecionando Salvar Cópia no menu Imagem

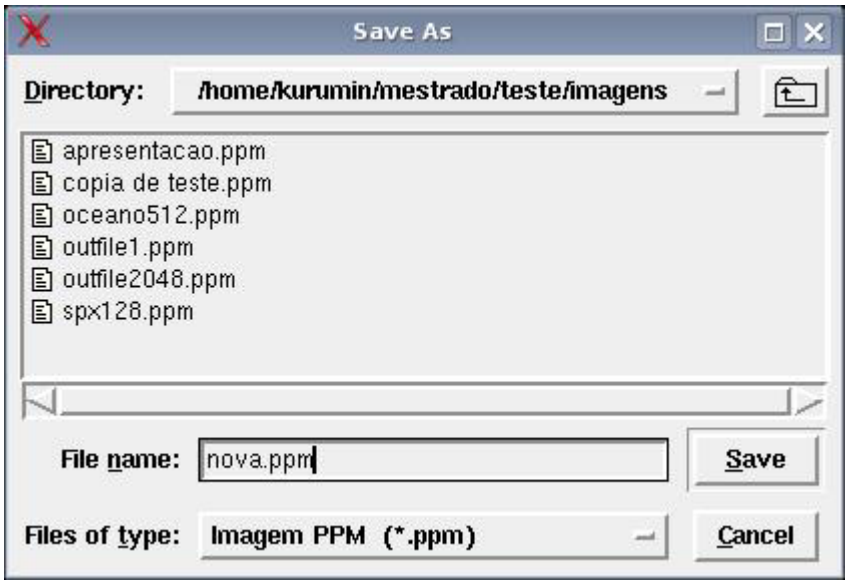

Figura 4.29: Janela para salvar cópia de imagem

CAPÍTULO 5
RESULTADOS EXPERIMENTAIS

Como ferramenta OceanView 3D foi desenvolvida para prover ao usuário imagens de dados oceânicos em tempo-real com a flexibilidade de se explorar o hardware disponível da melhor forma possível, apresentamos neste capítulo uma avaliação do desempenho da ferramenta tanto para a renderização local em hardware gráfico, como para a renderização paralela em cluster de PCs. Estamos interessados em mostrar que ambas as estratégias têm grande potencial para a geração de imagens em tempo-real e que, portanto, com a flexibilidade oferecida, caso uma estratégia não seja viável, a outra certamente fornecerá resultados satisfatórios.

5.1. Ambiente

As técnicas de renderização adotadas pelo nosso sistema exigem ambientes distintos para execução. Por este motivo, foram adotados 3 diferentes ambientes para realização dos testes de desempenho, sendo um para o algoritmo de hardware gráfico e dois para o algoritmo de renderização paralela.

5.1.1. Ambiente para a renderização com hardware gráfico

A implementação em hardware da renderização foi avaliada em um Intel Pentium IV 3.6 GHz, 2 GB RAM, com uma placa gráfica NVidia GeForce 6800 256MB e um barramento PCI Express 16x bus, do Laboratório de Computação Gráfica da COPPE/UFRJ.

5.1.2. Ambientes para a renderização paralela

O desempenho da renderização paralela foi avaliado em dois clusters diferentes. O primeiro cluster contém 16 nós de processamento, com processador Intel Pentium III de 800Mhz e 512MB de memória em cada nó.

Os nós estão conectados por meio de placas de rede Fast Ethernet de 100Mbps. Foi utilizado o sistema operacional Linux com kernel versão 2.4.20 e biblioteca de troca de mensagens MPI.

O segundo cluster utilizado é um cluster de SMPs, onde cada nó de processamento possui dois processadores que compartilham a mesma memória. O cluster contém 8 nós SMP (contabilizando um total de 16 processadores) do Laboratório de Computação Paralela da COPPE/UFRJ. Os nós estão interconectados por uma rede Ethernet Gigabit, onde cada nó consiste de 2 processadores AMD Opteron 2 GHz, compartilhando 1GB de memória RAM e executando o sistema operacional Linux, kernel 2.6.7.

Nosso objetivo em avaliar o desempenho de OceanView 3D em dois clusters diferentes é de mostrar o quanto a arquitetura pode influenciar ou não no desempenho da renderização paralela. O primeiro cluster é um cluster mais antigo, com processadores mais lentos e com apenas um único processador em cada nó. O segundo cluster é mais moderno e representa uma tendência atual em termos de clusters de PCs, o uso de nós SMPs.

5.2. Dado Oceânico

O dado utilizado em nossos experimentos é uma amostra do Golfo do México obtida do Navy Research Laboratory (NRL), gentilmente cedidas pelo Prof. Robert Morhead. A amostra tem resolução de 1 grau de latitude e longitude e contém informação sobre a velocidade do oceano em 6 níveis diferentes de profundidade. A malha tetraedral desta amostra de dado contém 2854K tetraedros e foi gerada pelo módulo de geração de malhas do sistema. As Figuras 5.1 e 5.2 mostram as imagens volumétricas produzidas para esta massa de dados de dois pontos de vista diferentes.

As porções de terra não aparecem na imagem, elas aparecem como “áreas negras”. As imagens foram geradas com mapas de cores diferentes para enfatizar porções diferentes do dado. Na Figura 5.1, o intervalo de velocidades é representado pela variação de laranja (para maiores velocidades) até azul escuro (para menores velocidades). Já na Figura 5.2, as cores variam de verde claro ao azul escuro para sobressaltar apenas a velocidade na costa.

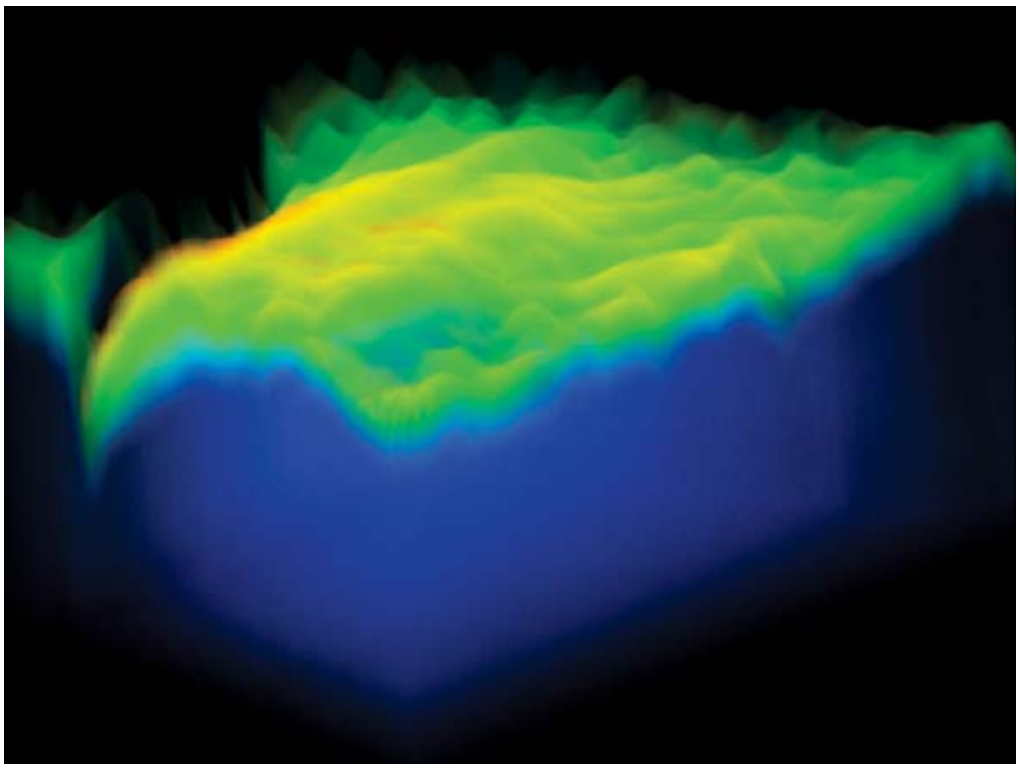

Figura 5.1: Imagem da velocidade do oceano no Golfo do México.

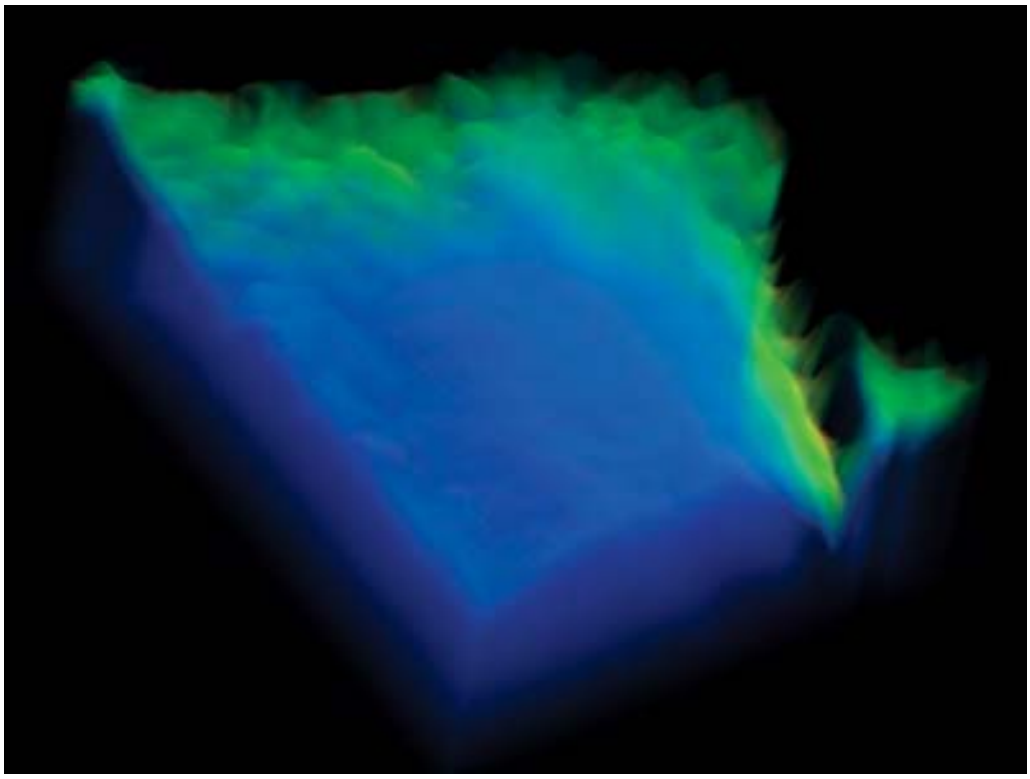

Figura 5.2: Imagem da velocidade do oceano no Golfo do México usando mapa de cor diferente.

5.3. Resultados com Placa Gráfica

A Tabela 5.1 mostra os resultados de tempo para o dado oceânico. Os tempos são dados em frames por segundo (frames/s) e tetraedros por segundo (tetraedros/s) usando uma imagem de 512×512 pixels e considerando que a imagem está constantemente sendo girada. Conforme podemos observar na tabela, a implementação da renderização em hardware atinge uma média de 26 frames por segundo, renderizando uma média de 1175K tetraedros por segundo. Esses resultados mostram que explorando-se a capacidade de programação das placas gráficas modernas, podemos obter tempos de renderização interativos para pequenos dados oceânicos. Infelizmente, dados oceânicos de maiores proporções (maiores que 256Mb) não poderão ser utilizados com esta técnica, pois os dados devem ser carregados totalmente na memória da placa gráfica para processamento. Para casos como este, o sistema OceanView 3D pode utilizar a renderização paralela.

Resultados de Tempo		
	Frames/s	Tetraedros/s
Dado NRL	26.3	1175K

Tabela 5.1: Resultados de tempo para a implementação em hardware.

5.4. Resultados do Sistema Paralelo

Os resultados da renderização paralela no cluster mais lento estão apresentados na Tabela 5.2 e no gráfico da Figura 5.3. Os resultados mostrados na Tabela 5.2 e na Figura 5.3, apresentam o tempo de execução, em segundos, da renderização paralela para 4, 8 e 16 processadores, quando utilizamos os três algoritmos diferentes de balanceamento de carga: *Nearest Neighbor* (NN), *Longest Queue* (LQ) e *Circular Distribution* (CD). Mostramos também o tempo em segundos quando a renderização paralela não apresenta nenhum balanceamento de carga.

Estes resultados reforçam os resultados obtidos em trabalhos anteriores [Coelho04b] e [Lopes06] em que o uso de mecanismos de balanceamento de carga é fundamental para o desempenho da renderização paralela. A execução sem balanceamento de carga ficou em torno de 73% mais lenta que as execuções que utilizaram balanceamento dinâmico. Em termos dos algoritmos de balanceamento, seguindo os resultados de

[Coelho04b], obtivemos resultados ligeiramente superiores para 16 processadores utilizando o algoritmo *Circular Distribution*.

	Algoritmos de Balanceamento			
# proc	Sem balanceamento	NN	LQ	CD
4	190,2	92,1	85,1	82,3
8	89,3	55,7	48,7	48,6
16	77,9	53,4	45,1	36,5

Tabela 5.2: Tempos de execução (em segundos) da renderização paralela no primeiro cluster.

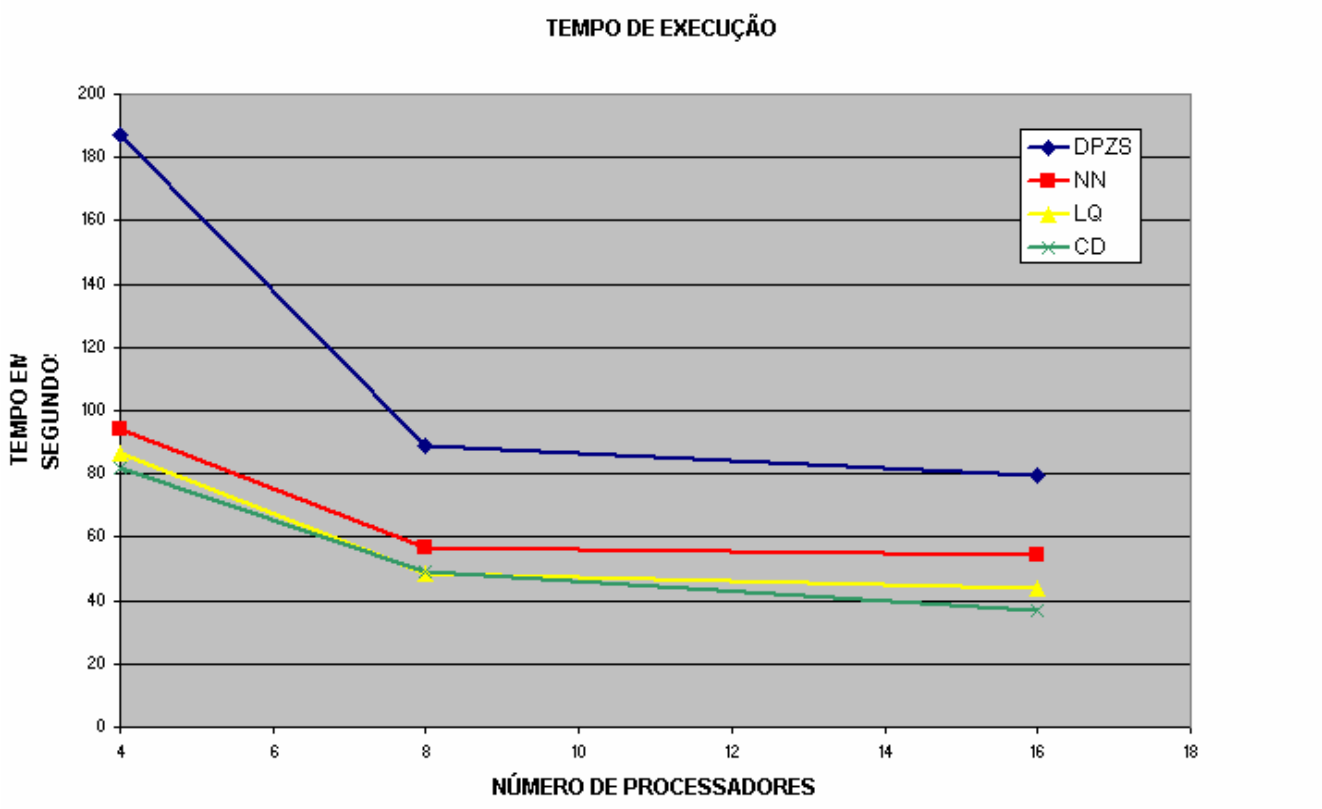

Figura 5.3: Gráfico de tempo de execução no primeiro cluster

Para efeito de comparação com os resultados obtidos em outra plataforma, cujo processador é bem rápido, apresentamos na Figura 5.4 o gráfico com o *speedup* obtido no primeiro cluster. O *speedup* representa a aceleração obtida com o processamento paralelo, pode ser definido como a relação entre o tempo gasto para executar uma tarefa com um processador e o tempo gasto com n processadores. A fórmula utilizada é:

$$S = T(1) / T(n)$$

Onde T(1) representa o tempo de execução serial quando o algoritmo executa em apenas 1 processador e T(n) representa o tempo da execução paralela, que representa o maior

tempo de execução dos n processadores. Por esta razão, o balanceamento de carga tem tamanha influência no *speedup* de uma aplicação, quanto mais desbalanceada estiver a carga, o processador que receber a maior carga, vai demorar mais para terminar.

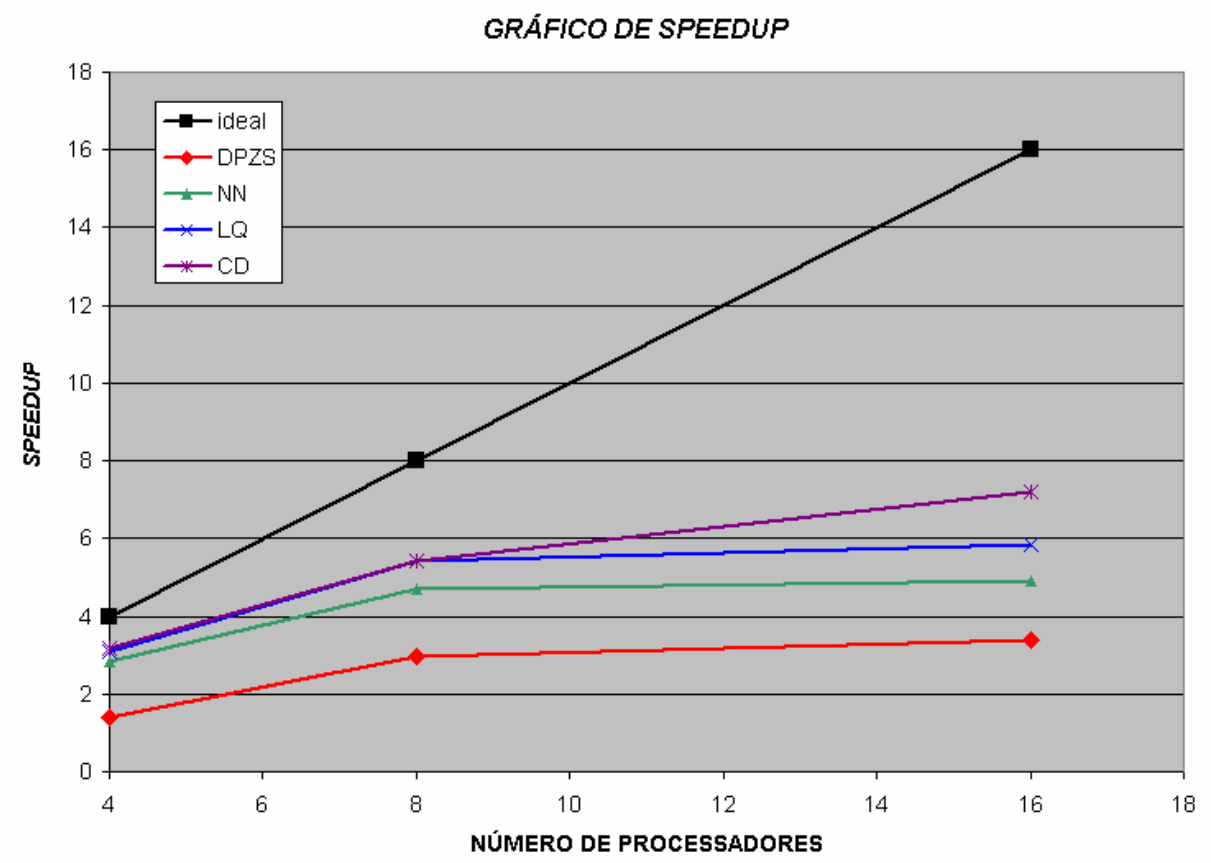

Figura 5.4: *Speedups* obtidos pelo sistema paralelo utilizando os métodos de balanceamento de carga primeiro cluster. T(1) = 263s.

Os *speedups* obtidos para a renderização paralela no primeiro cluster são baixos para a quantidade de processadores envolvidos. Estes resultados mostram apenas o potencial de aceleração que a paralelização da renderização pode prover. Entretanto, como a massa de dados é pequena e os processadores do cluster razoavelmente lentos, a aceleração ainda é baixa. Em outras palavras, neste cluster, para a massa de dados testada, não compensa utilizar mais do que 4 processadores para acelerar a renderização.

Embora tenhamos obtido *speedups* baixos no primeiro cluster, realizamos também experimentos no cluster mais moderno e mais veloz. Os resultados da renderização paralela no cluster de SMPs estão apresentados na Tabela 5.3 e no gráfico da Figura 5.5. A Tabela 5.3 apresenta os resultados de tempo de execução para 4, 8 e 16 processadores (que correspondem a 2, 4 e 8 nós de processamento do cluster), utilizando os três algoritmos

diferentes de balanceamento de carga e quando não utilizamos nenhum balanceamento de carga.

Conforme podemos observar na tabela, os tempos obtidos para o cluster de SMPs são bastante inferiores aos tempo obtidos para o primeiro cluster. A diferença do processador utilizado no cluster explica este fato. Em termos do algoritmo de balanceamento de carga utilizado, entretanto, obtivemos resultados diferentes com o cluster de SMPs. Neste cluster, o fato dos dois processadores de um mesmo nó SMP compartilharem memória explica os melhores resultados do algoritmo *Nearest Neighbor*. A troca de carga entre processadores de um mesmo nó SMP é bem mais barata.

Por este motivo, optamos por analisar o *speedup* obtido por DPZSweep neste cluster quando utilizamos os três algoritmos de balanceamento. O gráfico da Figura 5.5 mostra os *speedups* obtidos pelo sistema DPZSweep na visualização do dado oceânico, executando em 1, 2, 4 e 8 nós SMP (contendo respectivamente 2, 4, 8 e 16 processadores), utilizando as três estratégias de balanceamento (NN, LQ e CD) e sem utilizar balanceamento de carga (DPZ). Conforme podemos observar neste gráfico, a paralelização da renderização não é suficiente para se atingir bons *speedups*. A execução da renderização paralela sem o balanceamento de carga obteve bons *speedups* apenas para 2 processadores que pertecem ao mesmo nó SMP. Para mais processadores o desbalanceamento de carga diminui os benefícios da paralelização. Por outro lado, quando o balanceamento de carga *Nearest Neighbor* é utilizado, obtivemos excelentes resultados de *speedup*.

	Algoritmos de Balanceamento			
# proc	Sem balanceamento	NN	LQ	CD
4	43,5	31,0	32,5	31,1
8	40,3	16,3	20,4	19,5
16	35,1	8,9	11,3	10,8

Tabela 5.3: Tempos de execução (em segundos) da renderização paralela no cluster de SMPs.

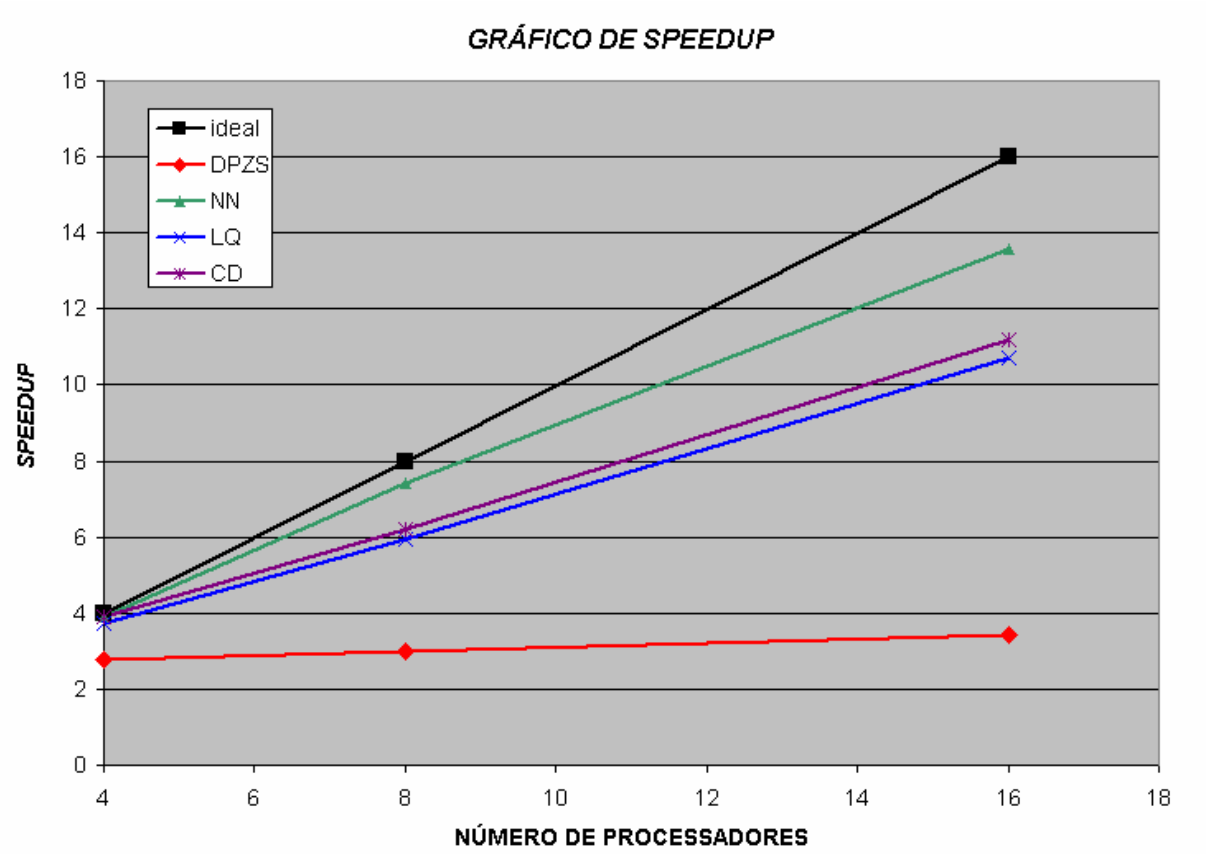

Figura 5.5: *Speedups* obtidos pelo sistema paralelo utilizando o método de balanceamento de carga no cluster SMP. T(1) = 121s.

Comparando a execução no primeiro cluster com a execução no segundo cluster, observamos que, conforme esperado, o uso de processadores mais velozes e de nós SMP melhorou bastante o desempenho da renderização paralela. Melhorou também o desempenho do balanceamento *Nearest Neighbor* com relação aos outros algoritmos de balanceamento. Os resultados de *speedup* mostram que a paralelização com balanceamento de carga tem potencial para acelerar a renderização. Entretanto, vale ressaltar que, nem sempre é possível obter tempos interativos com a solução paralela. Principalmente devido aos overheads de comunicação envolvidos.

Portanto, nossos resultados experimentais comprovam que usar o paralelismo é mais interessante para massas de dados muito grande, que não cabem na memória principal. Para essas massas de dados, o overhead da comunicação terá peso menor quanto mais imagens sob diferentes pontos de vista forem geradas. Com massas de dados pequenas, o uso de hardware gráfico é mais indicado.

CAPÍTULO 6
CONCLUSÕES

Neste trabalho, apresentamos uma ferramenta para visualização volumétrica eficiente de dados oceânicos em larga escala, chamada OceanView 3D. Visualizar a estrutura interna dos oceanos permite uma exploração mais dinâmica e detalhada de características como temperatura, velocidade, salinidade ou massa. Em outras palavras, converter dados numéricos em imagens, permite que os especialistas possam extrair de forma fácil e direta informações substanciais sobre fenômenos do oceano de difícil observação. Além disso, a visualização volumétrica de oceanos tem grande contribuição em áreas como gerenciamento de pesca e de mamíferos, mineração e indústria petrolífera, e pesquisas sobre o clima.

O sistema OceanView 3D foi concebido para ser uma ferramenta de fácil uso e que reúne numa única aplicação a possibilidade de explorar diferentes recursos de renderização volumétrica em alto desempenho. Exploramos o uso de hardware gráfico (GPU) e o uso de processamento paralelo em cluster de PCs. O sistema OceanView 3D possui, também, a capacidade de importar dados oceânicos de diversos formatos bem como a geração de seqüências de imagens que garantem maior dinamismo na visualização e análise das imagens.

Uma das principais vantagens do sistema OceanView 3D está na flexibilidade de uso da renderização de alto desempenho. Normalmente, as ferramentas de visualização volumétrica optam por atacar o problema de desempenho, na geração de imagens em tempo-real, de uma única forma, seja utilizando processamento paralelo, seja utilizando os benefícios da placa gráfica. Estamos defendendo neste trabalho que nenhuma das duas formas de se resolver o problema de desempenho é ideal para todas as situações. Quando o dado é maior que a capacidade da memória da GPU, o uso de um cluster com renderização paralela é mais indicado, caso contrário, o uso da placa gráfica é mais eficiente.

Além da exploração efetiva de mecanismos de alto desempenho, o sistema OceanView 3D permite também o armazenamento de imagens pré-geradas para uma futura visualização na forma de animação. Este armazenamento funciona como uma *cache* de imagens geradas. Utilizando esta *cache*, evitamos ter que renderizar um mesmo dado mais de uma vez, dado que já foi mostrado que a renderização é computacionalmente muito custosa.

Com uma interface gráfica bastante amigável, realizamos alguns experimentos com dados provenientes do Navy Research Laboratory. Nossos resultados experimentais mostraram que a proposta baseada no uso de hardware gráfico apresentou tempos interativos

para o dado do Golfo do México com 2854K tetraedros. Mostramos também que um cluster de SMPs é uma execelente plataforma para o sistema paralelo DPZSweep, especialmente para a estratégia de balanceamento de carga *Nearest Neighbor*. A curva de *speedup* apresentou comportamento quase linear. Além disso, nossos resultados experimentais comprovaram que usar o paralelismo é mais interessante para massas de dados muito grande, que não cabem na memória principal.

Apesar das duas soluções propostas aqui terem sido implementadas e avaliadas independentemente, ambas atingiram bons resultados de desempenho. Isto justifica a implementação do sistema OceanView 3D que integra essas duas soluções numa ferramenta de visualização de oceano que tira proveito de hardware gráfico ou cluster SMP, para tratar dados em larga escala.

O sistema OceanView 3D é um avanço em relação a disponibilização de recursos de visualização científica de oceanos eficiente e a baixo custo. Suas funcionalidades permitem um ganho substancial de produtividade por parte do especialista e permitem flexibilidade na escolha da fonte de dados oceânicos, bem como da melhor metodologia para o seu processamento.

Como trabalho futuro, pretendermos realizar estudos com outras massas de dados oceânicos, principalmente com massas de dados muito grandes. Pretendemos também estudar uma implementação do sistema em um cluster cujos nós possuam placas gráficas independentes, para que possamos explorar de forma integrada as vantagens das duas metodologias de renderização de alto desempenho utilizadas neste trabalho. Outra alternativa que pode ser explorada no futuro é a utilização de *grids* de computadores para o processamento. Em termos de melhorias na ferramenta, pretendemos realizar a escolha da estratégia de alto desempenho de forma automática, a partir de informações sobre a massa de dados e do hardware que a ferramenta está executando e a inclusão de visualização de informações geográficas, tais como: indicação de norte geográfico, longitude, latitude e profundidade, além das deformações causadas pelos sistemas de projeção da Terra.

REFERÊNCIAS BIBLIOGRÁFICAS

[ABRAGAMES04] ABRAGAMES - Associação Brasileira das Desenvolvedoras de Jogos Eletrônicos. Plano Diretor da Promoção da Indústria de Desenvolvimento de Jogos Eletrônicos no Brasil, p. 27, 21 de dezembro de 2004.

[Carvalho06] André C. P. de L. F. de Carvalho et al. Grandes Desafios da Pesquisa em Computação no Brasil – 2006 – 2016. Relatório sobre o Seminário realizado em São Paulo em 8 e 9 de maio de 2006.

[Challinger93] J. Challinger. Scalable parallel volume raycasting for nonrectilinear computational *grids*. In ACM SIGGRAPH Symposium on Parallel Rendering, pages 81--88, November 1993.

[Cirne01] W. Cirne, and K. Marzullo. Open Grid: A User-Centric Approach for Grid Computing. In 13th Symposium on Computer Architecture and High Performance Computing, September 2001.

[Coelho04a] COELHO, A., Balanceamento de Carga em Sistemas Paralelos de Visualização Volumétrica. Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Computação - área de concentração Geomática, Rio de Janeiro, 2004.

[Coelho04b] A. Coelho, M. Nascimento, C. Bentes, M.C. Castro, and R. Farias. Parallel volume rendering for ocean visualization in a cluster of pcs. In Geoinfo 2004, pages 291–304, 2004.

[Comba03] COMBA, J.L.D.; Dietrich, C.A.; Pagot, C.A. & Scheidegger, C.E. Computation on GPUs: from a programmable pipeline to an efficient stream processor. Revista de Informática Teórica e Aplicada, 2003, 10-1, 41-70

[Farias00] FARIAS, R.; MITCHELL, J.; SILVA, C., ZSWEEP: an Efficient and Exact Projection Algorithm For Unstructured Volume Rendering, In 2000 Volume Visualization Symposium, pp. 91-99, Outubro, 2000.

[Foster03] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid – Enabling Scalable Virtual Organizations. To appear in the International Journal of Supercomputing Applications.

[Foster98] I. Foster, and C.Kesselman. The Globus Project: A Status Report. In Heterogeneous Computing Workshop. IEEE Press, 1998.

[Franca05] FRANCA, Alexander, Tcl/Tk : programação Linux / Alexander Franca. – Rio de Janeiro: Brasport, 2005.

[Gary01] Gary Morris, Niushant Kaul, and Conrad Ali, Adel L.and Johnson. Employing mutual-exclusion algorithms for collaborative immersive visualization–cthru c: a case study. In Proc. SPIE, pages 96–103, 2001.

[Hofsetz00] C. Hofsetz and K.-L. Ma. Multi-threaded rendering unstructured-grid volume data on the sgi origin 2000. In Third Eurographics Workshop on Parallel Graphics and Visualization, 2000.

[Kaufman88] A. Kaufman and R. Bakalash. Memory and Processing Architecture for 3D Voxel-Based Imagery. In *IEEE Computer Graphics and Applications*, Vol. 8, No. 6, pages 10-23, November 1988.

[Liu05] H. Liu, Y. S.and Zhang, D. A. Yuen, and M. Wang. Highperformance computing and visualization of tsunamis and wind-driven waves. In American Geophysical Union, FallMeeting, 2005.

[Lopes06] LOPES, Andrei; BENTES, Cristiana; FARIAS, Ricardo. Visualização de Alto Desempenho de Dados oceânicos em Larga Escala. In: XXXII Conferência Latinoamericana de Informática - CLEI 2006, Santiago, 2006.

[Ma95] K.-L. Ma. "Parallel volume *ray-casting* for unstructured-grid data on distributed-memory architectures". IEEE Parallel Rendering Symposium, pags 23-30, Outubro 1995.

[Ma97] K.-L. Ma and T. Crockett. “A scalable parallel cell-projection volume rendering algorithm for three-dimensional unstructured data”. IEEE Parallel Rendering Symposium, pags 95-104, Novembro 1997.

[Ma00] Kwan-Liu Ma and David M. Camp. High performance visualization of time-varying volume data over a wide-area network status. In Supercomputing ’00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), 2000.

[Ma01] K-L Ma and S. Parker. Massively Parallel Software Rendering for Visualizing Large-Scale Data Sets. IEEE Computer Graphics and Applications, pp. 72-83 July/August 2001.

[Maximo06] A.Maximo, R.Marroquim, C. Esperanca, and R. Farias. Gpu based cell projection for interactive volume rendering. In to appear in Brazilian Symposium on Computer Graphics and Image Processing, 2006.

[Mueller99] K. Mueller, T. Möller, R. Crawfis. *Splatting Without the Blur*. In: IEEE Visualization 1999, San Francisco, CA. *Proceedings. . .* p. 363–370.

[Oparin38] Oparin, A. I. *The Origin of Life*. Nova York: Dover, 1952 (primeira publicação em 1938).

[Paiva99] PAIVA, A. C.; SEIXAS, R. B.; GATTASS, M. **Introdução à Visualização Volumétrica**. Departamento de Informática, PUC-Rio, Inf. MCC03/99, Rio de Janeiro, 1999.

[Rosa04] YVES PEREIRA SANTA ROSA – Aquecimento Global: Suas Origens e Perspectivas - UNIFEOB - Centro Universitário da Fundação de Ensino Octávio Bastos. São João da Boa Vista, SP, 2004. Online: Disponível na Internet via http://www.feob.br/novo/cursos/cbiologicas/monografias/2004/Yves%20Pereira%20Santa%20Rosa.pdf

[Rost86] ROST, Randi J., OFF – A 3D Object File Format, Digital Equipament Corporation. Workstation Systems Engineering. Palo Alto, Ca, 1986. Online: Disponível na Internet via http://www.martinreddy.net/gfx/3d/OFF.spec

[Schneider98] Bengt-Olaf Schneider. Parallel Rendering on PC Workstations. In International Conference on Parallel and Distributed Processing Techniques and Applications, 1998.

[Wikipédia07] Wikimedia Foundation - Wikipédia, a Enciclopédia Livre. Oceano. Online: disponível na Internet via http://pt.wikipedia.org/wiki/Oceano. Arquivo consultado em 12 de abril de 2007.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo